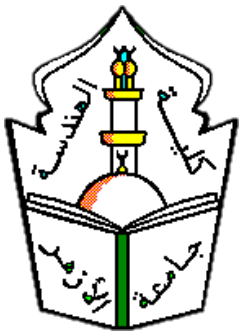
Al-Azhar University

Faculty of Engineering

Systems and Computers Engineering Department



Smart AI Library - Intelligent Book Recommendation System (Learnova)

Prepared by:

Reham Ehab El-Sayed Ahmed Barakat

Mariam Mohamed Abdullah Mohamed

Norhan Abdelrahman Fathy Salam

Hend Mohamed Ahmed Abdel Motaleb

Wafaa Yousry Ahmed Kohia

Salma Ibrahim Ali

Supervised by:

Dr. Sayed Noah

July 2026

Acknowledgments

In the name of Allah, the Most Gracious, the Most Merciful.

All praise and thanks are due to Allah, the Lord of the worlds, for His countless blessings and for granting us the strength and knowledge to complete this project. Without His guidance and mercy, none of this would have been possible.

We express our deepest gratitude to our beloved families for their unwavering support, prayers, and encouragement throughout our academic journey. Their love and faith have been a constant source of motivation and strength.

We are profoundly grateful to our esteemed supervisor, **Dr. Sayed Noah**, for his invaluable guidance, patience, and insightful feedback. His expertise and dedication were instrumental in shaping the quality and direction of this work.

We also extend our sincere thanks to our teachers and mentors who have imparted wisdom and knowledge throughout our studies. Their efforts have contributed significantly to our personal and academic growth.

May Allah reward everyone who supported us in this endeavor and bless them abundantly.

Alhamdulillah.

Abstract

In the contemporary digital learning landscape, the challenge of delivering personalized, accessible, and time-efficient education has become increasingly critical. Modern learners face a fundamental paradox: unprecedented access to knowledge paired with diminishing time to engage with it. This project presents **Learnova** — an AI-powered Audiobook and E-Learning platform designed to bridge the gap between fragmented audiobook services and structured educational environments through intelligent automation, audio-first delivery, and interactive course support.

The platform enables users to transform PDF books and educational materials into immersive, high-quality audio experiences via a state-of-the-art Text-to-Speech (TTS) pipeline, delivering a Spotify-inspired listening interface with persistent Mini Player support, adjustable playback speed, and chapter-level navigation. In parallel, Learnova provides a structured course management system featuring video lessons with AI-generated transcripts, enabling learners to search, review, and interact with course content through natural language. The `TranscribeAllLessons` pipeline automatically processes video lessons through an AI transcription service, making all course content fully searchable and accessible.

Beyond audio and course delivery, Learnova integrates a suite of six Python AI microservices covering the full intelligent library pipeline: semantic vectorisation using `all-mpnet-base-v2` embeddings stored in Pinecone, a FLAN-T5-based map-reduce summarisation service, a Retrieval-Augmented Generation (RAG) Q&A service powered by GPT-4o-mini with Server-Sent Events streaming and page-level citation, a hybrid recommendation engine combining Neural Collaborative Filtering (NCF) with content-based filtering, and a semantic search service with intent-aware re-ranking. A context-aware conversational chatbot provides on-demand assistance grounded in book content, ensuring each user receives accurate, citation-backed answers.

The system is built on a modern cross-platform mobile architecture using **Flutter** and **Dart** for the frontend with BLoC state management, backed by a **PHP Laravel 12** RESTful API and **MySQL 8.0** database. Laravel acts as a central AI proxy gateway, forwarding requests to the appropriate microservice and degrading gracefully on service unavailability. An administrative panel built with FilamentPHP 3 provides full content management, user role control, and AI metadata oversight. Secure authentication is

handled through JWT and Laravel Sanctum, with support for OTP-based password reset and social login via Google and Facebook OAuth.

This project demonstrates the practical feasibility of unifying audio-first consumption, structured course progression, and a full AI knowledge pipeline within a single cohesive platform, establishing Learnova as a foundational model for next-generation educational systems that prioritize accessibility, personalization, and learner autonomy.

Contents

Acknowledgments . . . . . **i**

Abstract . . . . . **ii**

1 Introduction **1**

1.1 Overview 2

1.2 Background of the Project 2

1.3 Target Users 3

1.4 Motivation 3

1.5 Expected Benefits 4

1.6 Problem Statement 4

1.7 Project Contributions 5

1.8 Objectives of the Project 6

1.9 Main Features 6

1.10 Scope and Limitations 7

1.10.1 Scope 7

1.10.2 Limitations 7

1.11 System Architecture Overview 8

1.12 Methodology 8

1.12.1 Main Phases 8

1.12.2 Tools and Technologies 9

1.13 Project Management 9

1.13.1 Timeline and Milestones 10

1.13.2 Team Roles and Responsibilities 10

Chapter Summary 10

2 Literature Review **12**

2.1 Overview 13

2.2 Existing E-Learning and Audiobook Platforms 13

2.2.1 Popular Audiobook Platforms 13

2.2.2 Popular E-Learning Platforms . . . 13
2.3 Comparison of Traditional and AI-Powered Platforms . . . 14
2.4 Applications of AI in Education . . . 14
2.5 Recommendation Systems in E-Learning . . . 14
2.6 Retrieval-Augmented Generation (RAG) and LangChain . . . 15
2.7 Text-to-Speech Technologies . . . 16
2.8 Flutter as a Cross-Platform Framework . . . 16
2.9 Related Works . . . 16
2.10 Limitations of Existing Systems . . . 17
Chapter Summary . . . 18
3 Backend Implementation . . . 19
3.1 Overview . . . 20
3.2 Objectives . . . 21
3.3 Scope . . . 22
3.3.1 Included . . . 22
3.3.2 Excluded . . . 23
3.4 System Architecture . . . 23
3.4.1 AI Proxy Architecture . . . 25
3.5 Database Design . . . 26
3.5.1 users . . . 27
3.5.2 books . . . 28
3.5.3 courses . . . 28
3.5.4 course_lessons . . . 29
3.5.5 reviews . . . 29
3.5.6 bookmarks and personal_shelves . . . 29
3.5.7 ai_metadata . . . 30
3.6 API Documentation . . . 30
3.6.1 Authentication Endpoints . . . 31
3.6.2 Books and Discovery Endpoints . . . 32
3.6.3 Courses Endpoints . . . 33
3.6.4 Favorites and Authors Endpoints . . . 34
3.6.5 Onboarding and Profile Endpoints . . . 35
3.6.6 AI Services Endpoints . . . 35
3.7 AI Microservices . . . 36
3.7.1 RAG Q&A Request Sequence . . . 36
3.7.2 Courses AI — TranscribeAllLessons . . . 37

3.7.3 AI Service Configuration . . . 38
3.8 Admin Panel (FilamentPHP 3) . . . 38
3.8.1 Library Resource Screens . . . 40
3.8.2 Community and Administration Screens . . . 40
3.8.3 Dashboard Analytics . . . 41
3.9 Testing . . . 42
3.9.1 Testing Tools . . . 42
3.9.2 Sample Test Cases . . . 43
3.10 Challenges Faced . . . 44
3.11 Future Work . . . 44
Chapter Summary . . . 45

4 Artificial Intelligence Integration . . . 46

4.1 Ingestion Service — Document Processing Pipeline . . . 48
4.1.1 Text Extraction . . . 48
4.1.2 Cleaning and Chunking . . . 48
4.1.3 Kafka Publishing and Status Tracking . . . 49
4.2 Embedding Service — Semantic Vectorisation . . . 50
4.2.1 Design Rationale . . . 50
4.2.2 Model: all-mpnet-base-v2 . . . 50
4.2.3 BookEncoder Implementation . . . 51
4.2.4 Kafka-to-Pinecone Pipeline & Fault Tolerance . . . 52
4.3 Search Service — Two-Stage Semantic Retrieval . . . 52
4.3.1 Two-Stage Pipeline . . . 53
4.3.2 Caching and Similar-Book Search . . . 54
4.4 Summarisation Service — FLAN-T5 Map-Reduce & Mind Maps . . . 54
4.4.1 Model: FLAN-T5-large . . . 54
4.4.2 Map-Reduce Pipeline . . . 55
4.4.3 Mind Map Generation . . . 56
4.5 RAG Service — Retrieval-Augmented Generation Q&A . . . 57
4.5.1 Architecture . . . 57
4.5.2 Retriever — Similarity Threshold Filtering . . . 58
4.5.3 Chain — Grounded Prompting & Token Budget . . . 59
4.5.4 Streaming — Server-Sent Events (SSE) . . . 59
4.6 Recommendation Service — Hybrid NCF + CBF . . . 60
4.6.1 Neural Collaborative Filtering (NCF) . . . 60
4.6.2 Content-Based Filtering (CBF) . . . 61
4.6.3 Hybrid Fusion and User Tiers . . . 61

4.7 Mobile Client Integration — Lesson Assistant and Lightweight Fallback Pathway . . . 62
4.7.1 Course Lesson Assistant — Whisper Transcription and Contextual Chat . . . 63
4.7.2 Book Chat and Summarisation — Fallback Pathway . . . 63
4.7.3 Shared Mobile Client Architecture . . . 64
4.7.4 Design Trade-offs . . . 65
4.8 Ethical Considerations . . . 65
4.8.1 Hallucination Mitigation . . . 65
4.8.2 Transparency and Source Attribution . . . 66
4.8.3 Content Scope and Copyright . . . 66
4.8.4 Model Bias and Limitations . . . 66
Chapter Summary . . . 66

5 Flutter Mobile Implementation . . . 69
5.1 Overview . . . 70
5.2 Home, Search & Profile Screens . . . 71
5.2.1 Home Screen . . . 71
5.2.2 Search Screen . . . 71
5.2.3 Profile Screen . . . 72
5.3 Authentication & Onboarding Screens . . . 72
5.3.1 Onboarding — Topic Preferences . . . 73
5.4 Book Detail Screen . . . 74
5.4.1 Objective . . . 74
5.4.2 UI/UX Implementation . . . 74
5.4.3 API Endpoints . . . 76
5.4.4 Request and Response Samples . . . 76
5.4.5 Models . . . 77
5.4.6 State Management . . . 77
5.4.7 Integration Steps . . . 78
5.4.8 Challenges and Solutions . . . 78
5.5 Author Profile Screen . . . 78
5.5.1 Feature Name . . . 78
5.5.2 Objective . . . 79
5.5.3 UI/UX Implementation . . . 79
5.5.4 API Endpoints . . . 80
5.5.5 Sample Response — GET /authors/{id}/profile (200 OK) . . . 80
5.5.6 Models . . . 81

5.5.7 State Management . . . 81
5.5.8 Key Difference from Instructor Profile . . . 81
5.6 Instructor Profile Screen . . . 81
5.6.1 Feature Name . . . 81
5.6.2 Objective . . . 81
5.6.3 UI/UX Implementation . . . 81
5.6.4 API Endpoints . . . 82
5.6.5 Models . . . 82
5.6.6 State Management . . . 82
5.6.7 Integration Steps . . . 82
5.6.8 Challenges and Solutions . . . 83
5.7 Course Detail Screen . . . 83
5.7.1 Feature Name . . . 83
5.7.2 Objective . . . 83
5.7.3 UI/UX Implementation . . . 83
5.7.4 API Endpoints . . . 84
5.7.5 Models . . . 85
5.7.6 State Management . . . 85
5.8 Media Player — Lesson and Book Audio Player . . . 85
5.8.1 Feature Name . . . 85
5.8.2 Objective . . . 85
5.8.3 UI/UX Implementation . . . 85
5.8.4 API Endpoints . . . 86
5.8.5 Mini Player Controller Architecture . . . 86
5.8.6 Challenges and Solutions . . . 88
5.9 Favourites Management System . . . 88
5.9.1 Feature Name . . . 88
5.9.2 Objective . . . 88
5.9.3 UI/UX Implementation . . . 88
5.9.4 API Endpoints . . . 89
5.9.5 Sample Response — GET /favorites/books (200 OK) . . . 89
5.9.6 FavoritesBloc State Design . . . 90
5.9.7 Integration Steps . . . 90
5.10 AI Chatbot — General and Contextual . . . 90
5.10.1 Objective . . . 90
5.10.2 UI/UX Implementation . . . 90
5.10.3 API Endpoints . . . 92
5.10.4 State Management . . . 92

5.10.5 Challenges and Solutions . . . 93
5.11 BLoC / Cubit State Machine . . . 93
5.12 REST API Endpoint Map . . . 94
5.13 Navigation and Routing (go_router) . . . 95
5.13.1 Key Routing Decisions . . . 96
5.14 Error Handling and UX Patterns . . . 96
5.15 Overall Challenges and Solutions . . . 98
Chapter Summary . . . 98

6 System Testing: Backend, AI Microservices & Flutter . . . 100
6.1 Overview . . . 101
6.2 Testing Objectives . . . 101
6.3 Testing Strategy and Levels . . . 101
6.4 Test Environment and Tools . . . 102
6.5 Backend (Laravel) Testing . . . 103
6.5.1 Unit and Feature Testing with PHPUnit . . . 103
6.5.2 Manual API Testing with Postman . . . 104
6.5.3 Postman Test Evidence . . . 105
6.6 AI Microservices Testing . . . 106
6.6.1 Example Test Case — RAG Hallucination Guard . . . 106
6.6.2 AI Service Health Checks . . . 107
6.7 Flutter Mobile Testing . . . 107
6.7.1 Widget and Cubit Testing . . . 107
6.7.2 Manual QA on Physical Devices . . . 107
6.8 Integration Testing . . . 108
6.9 Performance Testing . . . 109
6.10 Security Testing . . . 110
6.11 User Acceptance Testing (UAT) . . . 111
6.12 Test Results Summary . . . 112
6.13 Challenges Faced . . . 112
6.14 Future Testing Enhancements . . . 113
Chapter Summary . . . 113

List of Figures

3.1 Technology Stack Used in the Learnova Backend 20
3.2 End-to-End User Journey Across Authentication, Discovery, and AI Features 21
3.3 User Story Map Linking Each Actor to Their Core Needs and Value . . 22
3.4 Smart AI Library System Architecture 24
3.5 AI Proxy Request Routing Across Six Microservices 26
3.6 Entity Relationship Diagram of the Core Database Tables 27
3.7 API Request and Response Lifecycle for Two Representative Endpoints 31
3.8 Courses Module Flow for Students and Instructors 34
3.9 Sequence Diagram for the AI-Powered Semantic Search and Q&A Request 37
3.10 FilamentPHP Admin Panel Sections and Resources 39
3.11 Library Section — Resource List Views in the Admin Panel 40
3.12 Community Moderation and User Administration Screens 41
3.13 Live Admin Dashboard — Statistics Overview and Content Growth Trend 42

4.1 Overview of the AIGP Smart Library AI Integration Architecture . . . 47
4.2 Technology Stack Used in the AI Integration Layer 47
4.3 Ingestion Service — Document Processing Pipeline 49
4.4 Search Service — Two-Stage Semantic Retrieval Pipeline 54
4.5 Map-Reduce Summarisation Pipeline with Redis Cache and GPU Queue 56
4.6 RAG Q&A Sequence Diagram — from User Request to Streamed Answer 58
4.7 Recommendation Service — Hybrid NCF + CBF Fusion 62

5.1 Technology Stack Used in the Learnova Flutter Mobile App 70
5.2 Learnova Flutter Clean Architecture Layers 70
5.3 Home, Search, and Profile Screens 72
5.4 Authentication Screens: Login, Registration, and OTP-Based Password Recovery 73
5.5 OnboardingScreen — Topic Preference Selection Grid 74
5.6 BookDetailScreen Feature Flow Diagram 75
5.7 BookDetailScreen and its Audio Playback Interface 76
5.8 AuthorInfoScreen — Biography, Popular Releases, and Follow Button . 80
5.9 CourseInfoScreen — Preview, Summary, and Available Lessons 84

5.10 Media Player Screens for Video Lessons and Book Audio 86
5.11 Mini Player Global Overlay Architecture 87
5.12 Global FavoritesBloc Synchronisation Diagram 89
5.13 AI Chat & Summarise Request/Response Flow 91
5.14 AI Chatbot Surfaces: General Support, Contextual Book Chat, and AI Summary 91
5.15 BLoC/Cubit State Machine — BookDetailCubit Example 93
5.16 BLoC Event/State Flow: UI ↔ BLoC ↔ Data Layer 94
5.17 Learnova REST API Endpoint Map (Flutter Client) 94
5.18 go_router Navigation Structure 95
5.19 Error Handling & UX State Patterns 96

6.1 Postman Evidence — Registration and Login Endpoints 105
6.2 Postman Evidence — The Topics Onboarding Key-Mismatch Fix . . . 105
6.3 Postman Evidence — Course Content and Favourite Toggle Endpoints 105

List of Tables

1.1 Tools and Technologies Used in Learnova 9

2.1 Comparison of Traditional vs. AI-Powered E-Learning Platforms 14

3.1 System Architecture Layers and Responsibilities 25
3.2 users Table Schema 27
3.3 books Table Schema 28
3.4 courses Table Schema 28
3.5 course_lessons Table Schema 29
3.6 reviews Table Schema 29
3.7 bookmarks / personal_shelves Table Schema 29
3.8 ai_metadata Table Schema 30
3.9 Authentication API Endpoints 31
3.10 Books and Discovery API Endpoints 32
3.11 Courses API Endpoints 33
3.12 Favorites and Authors API Endpoints 34
3.13 Onboarding and Profile API Endpoints 35
3.14 AI Services API Endpoints 35
3.15 Python AI Microservices Overview 36
3.16 AI Service Configuration Parameters 38
3.17 Admin Panel Sections and Resources 39
3.18 Admin Panel Default Credentials 39
3.19 Dashboard Statistics 41
3.20 Testing Tools and Scope 42
3.21 Sample API Test Cases and Expected Results 43

4.1 Ingestion Service Chunking Parameters 49
4.2 Embedding Model Configuration and Design Decisions 51
4.3 Embedding Batch Job Processing Stages 52
4.4 Search Service Two-Stage Pipeline 53
4.5 FLAN-T5 Prompt Configurations by Summary Type 55
4.6 Mind Map Generation Pipeline with Fallback Strategy 57
4.7 RAG Service Component Overview 58

4.8 SSE Protocol Events in the RAG Streaming Endpoint 60
4.9 NCF Model Hyperparameters 61
4.10 Hybrid Fusion Alpha Values by User Tier 61
4.11 Lightweight Fallback Pathway — Endpoints and Backing Models 64
4.12 AI Services Comparison 68

5.1 Common Packages Used Across All Features 71
5.2 BookDetailScreen API Endpoints 76
5.3 BookDetailScreen Model Classes 77
5.4 BookDetailScreen Challenges and Solutions 78
5.5 AuthorInfoScreen API Endpoints 80
5.6 InstructorProfileScreen API Endpoints 82
5.7 InstructorProfileScreen Challenges and Solutions 83
5.8 CourseInfoScreen API Endpoints 84
5.9 Media Player API Endpoints 86
5.10 Media Player Challenges and Solutions 88
5.11 FavoritesScreen API Endpoints 89
5.12 AI Chatbot API Endpoints 92
5.13 AI Chatbot Challenges and Solutions 93
5.14 go_router Route Table 95
5.15 Error Handling and UX State Patterns 97
5.16 Overall Flutter Implementation Challenges and Solutions 98

6.1 Testing Levels Applied Across Learnova 102
6.2 Testing Tools by Layer 103
6.3 Test Environment Configuration 103
6.4 Sample Backend Test Cases and Results 104
6.5 AI Microservice Test Coverage 106
6.6 Sample Flutter Manual Test Cases 108
6.7 Verified End-to-End Integration Flows 109
6.8 Observed Response Times for Key Endpoints 110
6.9 Security Test Cases 110
6.10 UAT Scenarios and Outcomes 111
6.11 Overall Testing Summary 112

Chapter 1

Introduction

Contents

1.1 Overview 2
1.2 Background of the Project 2
1.3 Target Users 3
1.4 Motivation 3
1.5 Expected Benefits 4
1.6 Problem Statement 4
1.7 Project Contributions 5
1.8 Objectives of the Project 6
1.9 Main Features 6
1.10 Scope and Limitations 7
1.11 System Architecture Overview 8
1.12 Methodology 8
1.13 Project Management 9

1.1 Overview

The rapid growth of digital information and online learning resources has transformed the way people access knowledge. Thousands of books, research papers, and educational materials are now available in digital formats, providing unprecedented opportunities for learning and self-development. However, the increasing volume of available content has created new challenges in discovering relevant resources and extracting useful knowledge efficiently.

Learnova is an AI-powered Audiobook and E-Learning platform designed to enhance knowledge discovery and reading experiences. The system combines Artificial Intelligence technologies with digital library services to provide users with personalized book recommendations, semantic search capabilities, AI-generated summaries, and intelligent question-answering features.

Unlike traditional digital libraries that primarily focus on storing and retrieving books, Learnova acts as an intelligent reading assistant that helps users find relevant resources, understand content faster, and receive personalized learning experiences. Through its user-friendly mobile application and AI-powered modules, the platform aims to improve accessibility, efficiency, and engagement in digital reading and knowledge exploration.

This chapter introduces the project by discussing its background, target users, motivation, expected benefits, and the problem it aims to solve.

1.2 Background of the Project

Digital libraries have become essential tools for education, research, and knowledge management. Universities, schools, and organizations increasingly rely on digital repositories to provide users with easy access to books, journals, and learning materials. These systems have significantly improved information availability and accessibility compared to traditional physical libraries.

Despite these advancements, many digital library platforms still rely on conventional search mechanisms and static content organization methods. Users often struggle to locate relevant resources among thousands of available documents, especially when search results depend solely on keyword matching. As digital collections continue to expand, the process of finding appropriate books and learning materials becomes increasingly challenging.

At the same time, Artificial Intelligence technologies have revolutionized information retrieval, recommendation systems, and content analysis. AI-powered recommendation engines are widely used in modern platforms to personalize user experiences, while Natural Language Processing (NLP) techniques enable systems to understand content

meaning and provide more accurate search results.

The integration of AI capabilities into digital libraries creates opportunities for developing smarter and more personalized knowledge platforms. By combining recommendation systems, semantic search, summarization, and intelligent question-answering services, Learnova aims to address the limitations of traditional digital libraries and improve the overall reading experience.

1.3 Target Users

Learnova is designed to serve a wide range of users who interact with digital knowledge resources. The primary target users include:

- **Students** seeking quick access to academic books, educational materials, and learning resources.
- **Researchers** who require efficient methods for exploring large collections of documents and extracting relevant information.
- **Educators and instructors** looking for intelligent tools to support teaching and content discovery.
- **General readers** interested in exploring books and receiving personalized recommendations based on their interests.
- **Libraries and educational institutions** seeking modern digital transformation solutions that improve resource accessibility and user engagement.

The platform is designed to accommodate users with varying levels of technical expertise while providing advanced capabilities for academic and professional use.

1.4 Motivation

The development of Learnova is motivated by several challenges faced by users of traditional digital libraries and digital reading platforms.

First, the amount of available digital content continues to grow rapidly, making it difficult for users to discover books and resources that match their interests and needs. Traditional search systems often require users to spend significant time browsing and filtering large collections of content.

Second, most digital libraries rely heavily on keyword-based search methods that may not accurately understand user intent or the contextual meaning of documents. This limitation often leads to irrelevant search results and inefficient information retrieval.

Third, users frequently struggle to determine whether a book or document contains

the information they need without reading large portions of the content. This process can be time-consuming and discouraging.

Additionally, many digital libraries lack personalized recommendation systems that can guide users toward relevant resources based on their preferences, reading history, and interests. Recent advancements in Artificial Intelligence provide an opportunity to overcome these limitations through intelligent recommendations, semantic search, content summarization, and interactive question-answering capabilities.

1.5 Expected Benefits

The implementation of Learnova is expected to provide numerous benefits for users and educational organizations:

- Faster and more accurate discovery of books and learning resources.
- Personalized book recommendations based on user interests, preferences, and reading behavior.
- Improved information retrieval through semantic search technologies.
- Reduced reading time through AI-generated summaries.
- Enhanced understanding of content through intelligent question-answering features.
- Better organization and management of personal reading collections.
- Increased user engagement through personalized and interactive experiences.
- Improved accessibility to knowledge resources for students, researchers, and general readers.
- Support for digital transformation initiatives in educational institutions and libraries.
- A scalable platform capable of supporting future enhancements such as audiobooks and OCR services.

1.6 Problem Statement

Traditional digital libraries provide effective storage and access to digital resources; however, they often lack intelligent mechanisms for personalized content discovery, contextual understanding, and user-centered interaction. Most existing systems depend primarily on keyword-based search techniques, which may fail to capture the actual intent of users or the semantic meaning of documents.

As digital collections continue to grow, users face increasing difficulties in locating relevant books, identifying useful resources, and extracting valuable information efficiently. Furthermore, users often struggle to discover books that align with their interests and learning objectives due to the limited personalization capabilities available in traditional library systems.

The absence of advanced features such as semantic search, intelligent recommendations, automatic summarization, and direct question-answering further reduces the effectiveness of conventional digital libraries and limits user engagement. Consequently, users spend considerable time searching, filtering, and reviewing information manually, which negatively impacts productivity and learning efficiency.

Therefore, there is a need for an intelligent digital library platform that leverages Artificial Intelligence technologies to improve content discovery, resource recommendation, information retrieval, and user interaction. Learnova addresses this need by integrating recommendation systems, semantic search, AI-powered summarization, and intelligent question-answering capabilities into a unified digital knowledge platform.

1.7 Project Contributions

Learnova introduces several innovative contributions that enhance the functionality of traditional digital libraries and improve the overall reading experience:

- Development of an AI-powered digital library that combines intelligent book discovery with personalized reading assistance.
- Integration of a recommendation engine that suggests books and resources based on user preferences, reading history, ratings, and interests.
- Implementation of semantic search capabilities that understand the meaning and context of user queries rather than relying solely on keyword matching.
- AI-powered book summarization that enables users to quickly understand the main ideas and key concepts of books.
- Direct Question-and-Answer functionality that allows users to interact with book content and obtain relevant answers efficiently.
- Development of a modern cross-platform mobile application using **Flutter**, ensuring accessibility across Android and iOS devices.
- Creation of a personalized reading environment through favorites, bookmarks, notes, and personal library management features.
- Transformation of traditional digital libraries into intelligent reading assistants that support knowledge discovery and lifelong learning.

1.8 Objectives of the Project

The primary objective of Learnova is to develop an intelligent digital library platform that enhances knowledge discovery, content understanding, and user engagement through Artificial Intelligence technologies.

The specific objectives of the project are:

- Provide users with centralized and intelligent access to books, references, and educational resources.
- Implement an AI-driven recommendation system based on user preferences and reading behavior.
- Support semantic search and advanced filtering techniques beyond traditional keyword matching.
- Enable users to create personal reading shelves, bookmarks, notes, and favorites.
- Generate AI-powered summaries that reduce reading time and improve content understanding.
- Provide direct question-answering capabilities from book content.
- Deliver a clean, modern, and user-friendly mobile experience.
- Ensure system security, scalability, and performance for future growth and feature expansion.

1.9 Main Features

Learnova provides a comprehensive set of intelligent features designed to improve the reading and knowledge discovery experience:

- AI-Powered Book Recommendation System.
- Semantic Search Engine.
- AI-Based Book Summarization.
- Direct Question & Answer from Book Content.
- Personal Library Management.
- Favorites and Bookmarking System.
- Note-Taking and Highlighting Features.
- Smart Notifications and Personalized Suggestions.
- Offline Access to Saved Books and Notes.

- User Authentication and Profile Management.
- Responsive Cross-Platform Mobile Application (Flutter).
- Administrative Dashboard for Content Management.

1.10 Scope and Limitations

1.10.1 Scope

The scope of Learnova focuses on developing an intelligent digital library platform that combines traditional library services with Artificial Intelligence technologies. The system includes:

- Digital book and resource management.
- AI-powered recommendation services.
- Semantic search functionality.
- Book summarization services.
- Question-answering from book content.
- Personal reading libraries.
- Bookmarks, notes, and favorites management.
- Smart notification services.
- Mobile application support for Android and iOS.
- Administrative management tools.

1.10.2 Limitations

Despite its capabilities, the system has several limitations:

- AI-generated summaries and answers depend on the quality and availability of book content.
- Recommendation accuracy improves over time as more user interaction data becomes available.
- Semantic search effectiveness depends on the quality of indexed resources and NLP models.
- Offline mode is limited to previously saved books and notes.
- The current version focuses on digital resources and does not manage physical library inventories.

- Features such as OCR and extended Audiobook capabilities are planned as future enhancements.

1.11 System Architecture Overview

Learnova follows a modular and scalable architecture consisting of four major layers:

Frontend Layer (Flutter/Dart) Provides a responsive and user-friendly mobile interface for Android and iOS devices. It handles user interaction, navigation, content presentation, and audio playback using BLoC state management.

Backend Layer (Laravel/PHP) Manages business logic, authentication via Laravel Sanctum, RESTful API services, and communication between system components.

Database Layer (MySQL) Stores user information, preferences, reading history, book metadata, and application data. Ensures efficient data retrieval and management.

AI Processing Layer (Python / LangChain) Implements recommendation systems, semantic search, book summarization, and intelligent question-answering functionalities using OpenAI and Gemini APIs via a RAG pipeline.

Together, these layers create a scalable and maintainable architecture that supports efficient communication between users, library resources, and AI services.

1.12 Methodology

Learnova follows the **Agile Software Development Methodology**. Agile promotes iterative development, continuous improvement, and effective collaboration among team members throughout the project lifecycle.

1.12.1 Main Phases

Phase 1: Planning and Analysis

- Requirements gathering and analysis.
- Identifying system objectives and user needs.
- Defining project scope and specifications.

Phase 2: System Design

- Designing system architecture.
- Creating database schemas and ER diagrams.

- Developing UI/UX wireframes using Figma.

Phase 3: Development

- Building the Flutter mobile application.
- Developing Laravel backend APIs and services.
- Implementing authentication and user management.

Phase 4: AI Integration

- Developing recommendation models.
- Implementing semantic search functionality.
- Integrating summarization and question-answering modules.

Phase 5: Testing

- Unit Testing.
- Integration Testing.
- System Testing.
- Performance and Security Testing.

Phase 6: Deployment and Maintenance

- Deploying backend services.
- Publishing mobile applications.
- Monitoring system performance and implementing future improvements.

1.12.2 Tools and Technologies

Table 1.1. Tools and Technologies Used in Learnova

Category	**Technologies**
Frontend	Flutter (Dart), BLoC State Management
Backend	PHP Laravel, RESTful APIs, Laravel Sanctum
Database	MySQL
AI & ML	Python, LangChain, OpenAI API, Gemini API, RAG Pipeline
Dev & Deployment	Git & GitHub, Postman, Figma

1.13 Project Management

Effective project management practices are applied to ensure successful development, efficient collaboration, and timely project completion.

1.13.1 Timeline and Milestones

The project is divided into several major milestones:

- Requirements Analysis and Planning.
- System Design and Database Modeling.
- Mobile Application Development.
- Backend API Development.
- AI Module Development and Integration.
- System Integration and Testing.
- Documentation Preparation.
- Final Deployment and Presentation — *July 2026*.

1.13.2 Team Roles and Responsibilities

The project team consists of six members organized into three development groups:

Flutter Development Team (2 Members)

- Develop the mobile application UI and screens.
- Implement BLoC state management and navigation.
- Integrate frontend with backend APIs.

Backend Development Team (2 Members)

- Develop RESTful APIs using Laravel.
- Design and manage MySQL database structures.
- Implement authentication and authorization services.

AI Integration Team (2 Members)

- Develop the recommendation engine.
- Implement semantic search and RAG pipeline.
- Build book summarization and question-answering services.

Chapter Summary

This chapter introduced the Learnova project and discussed its background, target users, motivation, expected benefits, and problem statement. It also presented the project's objectives, contributions, main features, scope, system architecture, development methodology, and project management plan. By integrating Artificial Intelligence

technologies with digital library services, Learnova aims to transform traditional digital libraries into intelligent reading assistants that provide personalized recommendations, efficient content discovery, and enhanced user experiences. The next chapter reviews related work, existing systems, and technologies relevant to the proposed solution.

Chapter 2

Literature Review

Contents

2.1 Overview 13
2.2 Existing E-Learning and Audiobook Platforms 13
2.3 Comparison of Traditional and AI-Powered Platforms 14
2.4 Applications of AI in Education 14
2.5 Recommendation Systems in E-Learning 14
2.6 Retrieval-Augmented Generation (RAG) and LangChain 15
2.7 Text-to-Speech Technologies 16
2.8 Flutter as a Cross-Platform Framework 16
2.9 Related Works 16
2.10 Limitations of Existing Systems 17

2.1 Overview

This chapter presents a comprehensive review of the existing literature and related technologies relevant to the Learnova platform. The review covers existing e-learning and audiobook platforms, applications of Artificial Intelligence in education, recommendation systems, Retrieval-Augmented Generation (RAG) pipelines, Text-to-Speech technologies, and cross-platform mobile development using Flutter. The purpose of this review is to identify the limitations of current systems and justify the design decisions adopted in Learnova.

2.2 Existing E-Learning and Audiobook Platforms

The digital education and audiobook market has grown significantly over the past decade, giving rise to numerous platforms that serve learners worldwide. Understanding these platforms is essential to identifying gaps that Learnova aims to fill.

2.2.1 Popular Audiobook Platforms

Audible (Amazon) is the world's largest audiobook platform, offering over 500,000 titles with professional narration. While it provides a rich audio experience, it lacks intelligent content summarization, AI-powered Q&A, or personalized learning path features.

Spotify has expanded into audiobooks, integrating them alongside podcasts and music. Its recommendation engine is based on listening behavior, but it does not support educational features such as note-taking, bookmarks, or AI-generated summaries.

Google Play Books offers TTS functionality for e-books, allowing users to listen to any purchased book. However, the TTS quality relies on device-native engines and lacks contextual AI interaction with content.

Kobo provides e-reading and limited audiobook support, with basic annotation tools. It does not incorporate AI-driven recommendations or question-answering from book content.

2.2.2 Popular E-Learning Platforms

Coursera and **edX** offer structured online courses from universities worldwide. While they include video lectures and quizzes, they do not provide audiobook-style consumption or real-time AI interaction with course materials.

Khan Academy provides free educational content with adaptive exercises but does not support digital book management or AI-powered summarization.

2.3 Comparison of Traditional and AI-Powered Platforms

Traditional digital library and e-learning platforms primarily rely on keyword-based search and static content organization. Users are expected to manually browse large collections to locate relevant resources, which is time-consuming and often ineffective.

AI-powered platforms, by contrast, leverage machine learning models to understand user intent, personalize content delivery, and provide intelligent assistance. Table 2.1 summarizes the key differences between traditional and AI-powered platforms.

Table 2.1. Comparison of Traditional vs. AI-Powered E-Learning Platforms

Feature	**Traditional Platforms**	**AI-Powered Platforms**
Search	Keyword-based	Semantic / NLP-based
Recommendations	Manual / Category-based	Personalized ML models
Content Interaction	Read/Watch only	Q&A, Summarization, Chat
Audio Delivery	Limited / None	TTS with AI narration
User Adaptation	Static	Dynamic, behavior-driven
Personalization	Low	High

2.4 Applications of AI in Education

Artificial Intelligence has increasingly been applied in educational settings to enhance learning outcomes and improve user engagement. Tiwari [1] highlights that the integration of AI and machine learning in education has significant potential to personalize student learning experiences and improve academic performance.

Large Language Models (LLMs) such as GPT-4 and Gemini have opened new possibilities for educational tools. These models can generate explanations, answer questions, and summarize content with human-like fluency. Jayaram [2] discusses how AI-driven personalization can create hyper-personalized learning journeys tailored to each student type.

Natural Language Processing (NLP) techniques further enable systems to analyze the semantic meaning of user queries, going beyond simple keyword matching to understand context and intent.

2.5 Recommendation Systems in E-Learning

Recommendation systems are a critical component of modern e-learning platforms. da Silva et al. [3] conducted a systematic literature review on educational recommender systems and concluded that hybrid systems integrating collaborative filtering, content-

based approaches, and learner modeling demonstrate improved effectiveness in terms of learner engagement and outcomes.

A comprehensive review of recommender systems published in 2024 [4] synthesized recent advancements, including the increased use of deep learning architectures, graph-based models, and hybrid recommendation pipelines across domains including education.

Yanes et al. [5] proposed a machine learning-based recommender system for improving student learning experiences, achieving significant improvements in recommendation relevance through collaborative and content-based hybrid approaches.

In the context of Learnova, the recommendation engine analyzes user reading history, listening behavior, preferences, and ratings to suggest relevant books and educational materials.

2.6 Retrieval-Augmented Generation (RAG) and LangChain

Retrieval-Augmented Generation (RAG) is an AI framework that combines the generative capabilities of large language models with information retrieval from external knowledge bases. Instead of relying solely on pre-trained knowledge, RAG systems retrieve relevant documents at inference time and use them to ground the model's responses.

EL Maazouzi et al. [6] demonstrated the effectiveness of synergistically integrating LangChain, GPT models, and RAG for optimizing recommendation systems in e-learning environments. Their work confirms that RAG pipelines significantly improve the relevance and accuracy of AI-generated responses in educational contexts.

LangChain, released in 2022, is a versatile framework designed for building applications that leverage large language models [7]. It simplifies the integration of LLMs with external data sources, enabling developers to create RAG systems that combine generative models with retrieval systems. LangChain provides a robust API for connecting LLMs to vector databases, document stores, and external APIs.

Burgan et al. [8] developed a RAG chatbot application using LangChain and adaptive LLMs to assist university students in navigating institutional information, demonstrating the practical feasibility of LangChain-based RAG systems in higher education settings.

In Learnova, the RAG pipeline processes uploaded PDF books, chunks them into searchable segments, stores them in a vector database, and retrieves relevant passages to answer user questions with contextual accuracy.

2.7 Text-to-Speech Technologies

Text-to-Speech (TTS) technology converts written text into spoken audio, enabling audio-first consumption of digital content. Modern TTS systems powered by deep learning produce natural-sounding speech that closely resembles human narration.

In the context of mobile applications, TTS has become increasingly accessible through APIs and cross-platform frameworks. The `flutter_tts` package provides a Flutter plugin that leverages native TTS engines on Android (`TextToSpeech`) and iOS (`AVSpeechSynthesizer`), enabling multilingual speech synthesis with configurable parameters such as pitch, rate, and volume [9].

Commercial TTS APIs such as Google Text-to-Speech, Amazon Polly, and ElevenLabs offer high-quality neural voices suitable for audiobook-style delivery. These APIs support multiple languages, voice styles, and SSML (Speech Synthesis Markup Language) for fine-grained control over pronunciation and intonation.

Learnova integrates a TTS pipeline that converts PDF book content into high-quality audio, delivering a Spotify-inspired listening experience with playback speed control, chapter navigation, and a persistent Mini Player.

2.8 Flutter as a Cross-Platform Framework

Flutter is an open-source UI framework developed by Google that enables the creation of natively compiled applications for Android, iOS, web, and desktop from a single codebase using the Dart programming language.

Flutter's widget-based architecture and the BLoC (Business Logic Component) state management pattern provide a clean separation between UI and business logic, making it well-suited for complex, data-driven applications such as Learnova. The BLoC pattern ensures predictable state transitions and facilitates testability.

The availability of packages such as `just_audio` for audio playback, `flutter_tts` for TTS, and `dio` for HTTP communication makes Flutter particularly effective for building feature-rich audiobook and e-learning applications.

2.9 Related Works

Several prior works are directly relevant to the design and implementation of Learnova:

RamChat [8]: A RAG-based chatbot developed at Shepherd University using LangChain and LLMs to assist students with institutional information. This work validates the use of LangChain-based RAG pipelines in educational environments, a core component

of Learnova's Q&A feature.

Personalized E-Learning Recommender System Based on Autoencoders [10]: This study proposed using deep autoencoder networks to address the sparsity problem in collaborative filtering for e-learning recommendation, achieving improved personalization performance.

A Digital Recommendation System for Personalized Learning [11]: A review published in IEEE examining 60 articles on e-learning recommendation systems, identifying collaborative and content-based approaches as dominant methods with a recent shift toward machine learning.

AI-Driven Solutions for Library User Experiences [12]: This work explored how AI can improve library services through personalized knowledge dissemination and inclusive user experiences, directly informing Learnova's intelligent library management approach.

2.10 Limitations of Existing Systems

Based on the literature review, the following limitations are identified in existing systems:

- **Lack of audio-first learning**: Most digital library and e-learning platforms do not provide high-quality, AI-narrated audiobook experiences integrated with learning tools.
- **Limited AI interaction**: Existing audiobook platforms such as Audible and Spotify do not allow users to interact with book content through Q&A or summarization.
- **Weak personalization**: Traditional platforms rely on category-based browsing rather than behavior-driven recommendation engines tailored to individual learning patterns.
- **No RAG-based Q&A from books**: Current systems do not support contextual question-answering directly from uploaded educational materials using RAG pipelines.
- **Fragmented experience**: Audio consumption, reading, annotation, and AI assistance are spread across multiple disconnected applications, creating friction in the learning workflow.

Learnova addresses these limitations by unifying audio-first consumption, AI-powered interaction, personalized recommendations, and learning tools within a single, coherent mobile platform.

Chapter Summary

This chapter reviewed the existing literature on e-learning and audiobook platforms, AI applications in education, recommendation systems, RAG pipelines, TTS technologies, and cross-platform development with Flutter. The review identified key limitations in current systems, including the absence of integrated audio-first learning, limited AI interaction with book content, and weak personalization capabilities. These findings provide the foundation for the design decisions adopted in Learnova, which is presented in the following chapters.

Chapter 3

Backend Implementation

Contents

4.1 Overview 20
4.2 Objectives 21
4.3 Scope 22
4.4 System Architecture 23
4.5 Database Design 26
4.6 API Documentation 30
4.7 AI Microservices 36
4.8 Admin Panel (FilamentPHP 3) 38
4.9 Testing 42
4.10 Challenges Faced 44
4.11 Future Work 44

Laravel MySQL Python Docker Kafka Filament JWT

Figure 3.1. Technology Stack Used in the Learnova Backend

3.1 Overview

The modern reader, student, and researcher face a growing challenge: the sheer volume of available books and digital resources makes it increasingly difficult to discover content that truly matches one's interests, academic needs, or research goals. Traditional library systems rely on keyword-based search, which often returns irrelevant results and fails to understand the user's intent. There is also a lack of platforms that combine book management, educational courses, intelligent recommendations, and AI-powered features in one unified system.

Learnova addresses these challenges by building a centralized, scalable, and AI-integrated backend that powers personalized book and course discovery, semantic search, book summarization, and direct Q&A from book content — all through a clean RESTful API consumed by the Flutter mobile application. Figure 3.2 illustrates the complete journey of a user through the platform, from registration to the AI-powered features available on each book.

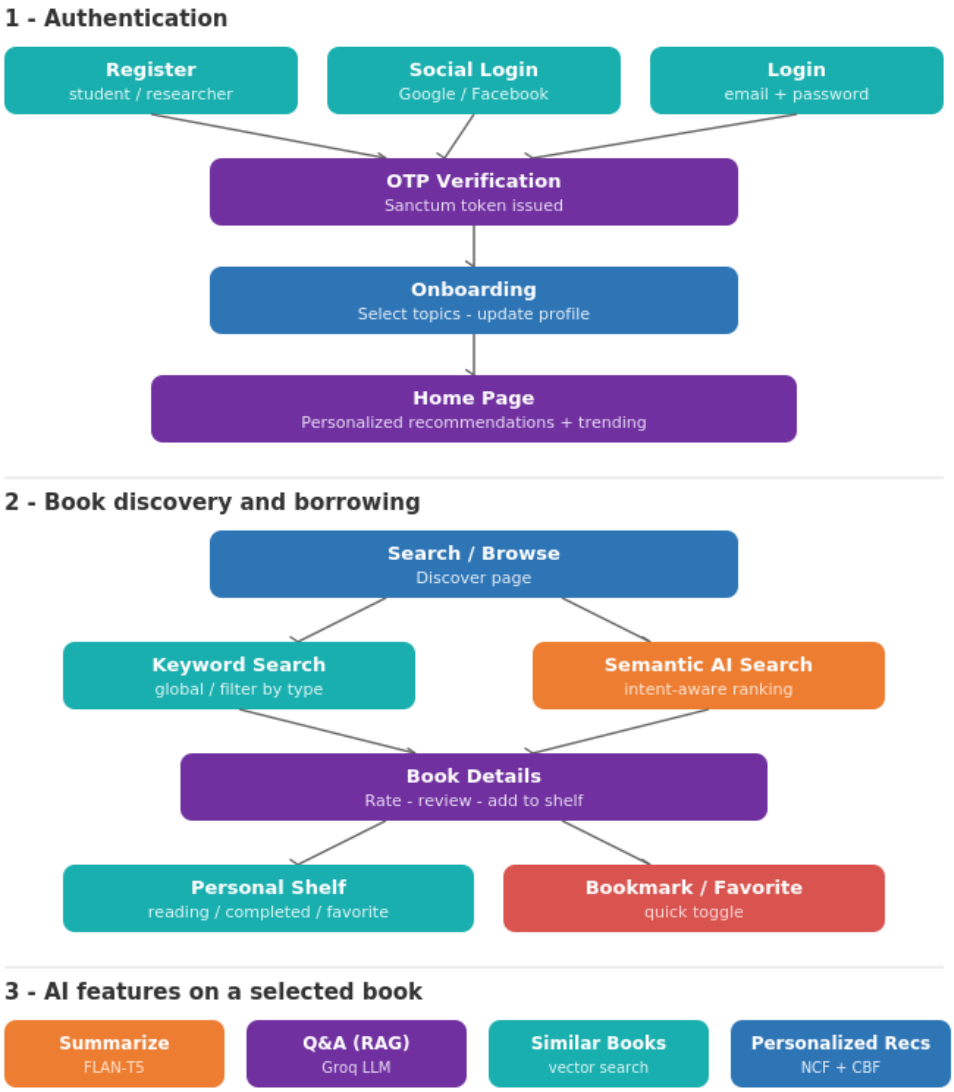


Figure 3.2. End-to-End User Journey Across Authentication, Discovery, and AI Features

3.2 Objectives

The main objectives of the backend system are:

- To provide a secure RESTful API consumed by the Flutter mobile application.
- To implement full CRUD operations for books, courses, authors, categories, and users.
- To integrate JWT-based authentication and role-based access control (Admin, Researcher, Student).
- To support a review and rating system allowing users to evaluate books.
- To manage user bookmark collections, personal reading shelves, and favorites.
- To integrate 6 Python AI microservices covering ingestion, recommendations,

semantic search, summarization, and Q&A.

- To support course management with video lessons and AI-powered lesson transcription.
- To deliver an admin panel (FilamentPHP 3) for efficient management of all platform resources.
- To implement OTP-based password reset and social login (Google/Facebook).

Figure 3.3 maps these objectives back to the platform's three actors — Student, Researcher, and Admin — showing the core user stories that drove the design of the API, the AI proxy layer, and the admin panel.

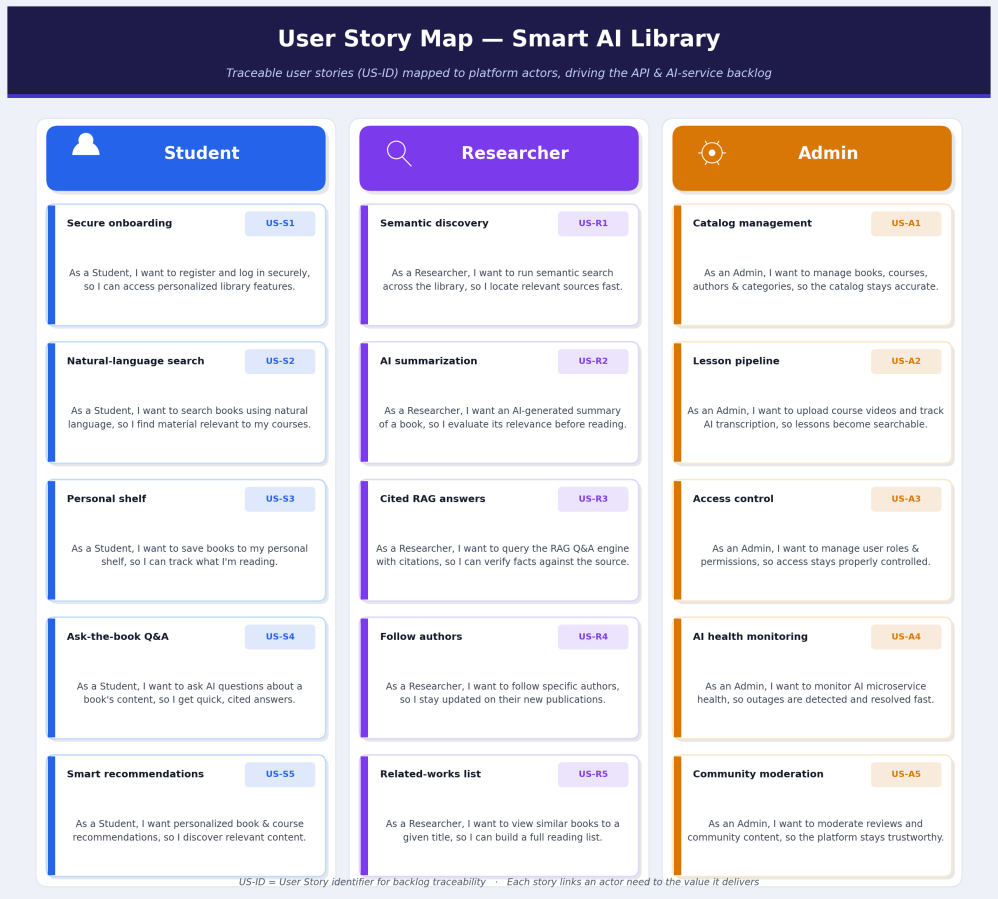

User Story Map — Smart AI Library
Traceable user stories (US-ID) mapped to platform actors, driving the API & AI-service backlog
Student
Secure onboarding
US-S1
As a Student, I want to register and log in securely, so I can access personalized library features.
Natural-language search
US-S2
As a Student, I want to search books using natural language, so I find material relevant to my courses.
Personal shelf
US-S3
As a Student, I want to save books to my personal shelf, so I can track what I'm reading.
Ask-the-book Q&A
US-S4
As a Student, I want to ask AI questions about a book's content, so I get quick, cited answers.
Smart recommendations
US-S5
As a Student, I want personalized book & course recommendations, so I discover relevant content.
Researcher
Semantic discovery
US-R1
As a Researcher, I want to run semantic search across the library, so I locate relevant sources fast.
AI summarization
US-R2
As a Researcher, I want an AI-generated summary of a book, so I evaluate its relevance before reading.
Cited RAG answers
US-R3
As a Researcher, I want to query the RAG Q&A engine with citations, so I can verify facts against the source.
Follow authors
US-R4
As a Researcher, I want to follow specific authors, so I stay updated on their new publications.
Related-works list
US-R5
As a Researcher, I want to view similar books to a given title, so I can build a full reading list.
Admin
Catalog management
US-A1
As an Admin, I want to manage books, courses, authors & categories, so the catalog stays accurate.
Lesson pipeline
US-A2
As an Admin, I want to upload course videos and track AI transcription, so lessons become searchable.
Access control
US-A3
As an Admin, I want to manage user roles & permissions, so access stays properly controlled.
AI health monitoring
US-A4
As an Admin, I want to monitor AI microservice health, so outages are detected and resolved fast.
Community moderation
US-A5
As an Admin, I want to moderate reviews and community content, so the platform stays trustworthy.
US-ID = User Story identifier for backlog traceability · Each story links an actor need to the value it delivers

Figure 3.3. User Story Map Linking Each Actor to Their Core Needs and Value

3.3 Scope

3.3.1 Included

- User registration, login, OTP password reset, and social login (Google/Facebook).
- JWT-based authentication via Laravel Sanctum with role-based access control

(admin, researcher, student).

- Full CRUD for Books, Courses, Course Videos, Authors, Categories, and Chapters.
- Advanced search and filtering: global search, semantic search, search history, and trending.
- Review and rating system for books.
- Bookmark, favorites, and personal library shelves management.
- Onboarding: user topic preferences and profile setup.
- Admin panel (FilamentPHP 3) with full CRUD, file uploads, and analytics dashboard.
- Full AI integration: 6 Python microservices via Docker (ingestion, embedding, recommendations, semantic search, summarization, RAG Q&A).
- AI-powered lesson transcription via `TranscribeAllLessons` command.
- Proxy architecture: Laravel acts as a gateway forwarding requests to AI microservices.

3.3.2 Excluded

- Video conferencing or live streaming functionality.
- Payment processing and subscription management.
- Direct instructor consultation chat.
- OCR for scanned documents (planned future enhancement).
- Audio-book streaming (planned future enhancement).

3.4 System Architecture

The Learnova backend follows a layered RESTful API architecture built on **Laravel 12**. The system is designed around a proxy pattern where Laravel serves as the central gateway: it handles authentication, business logic, and database operations, while forwarding AI-related requests to dedicated Python microservices running in Docker containers. Figure 3.4 shows the overall layered architecture, illustrating how the Flutter mobile app, Laravel backend, admin panel, database, and AI microservices interact.

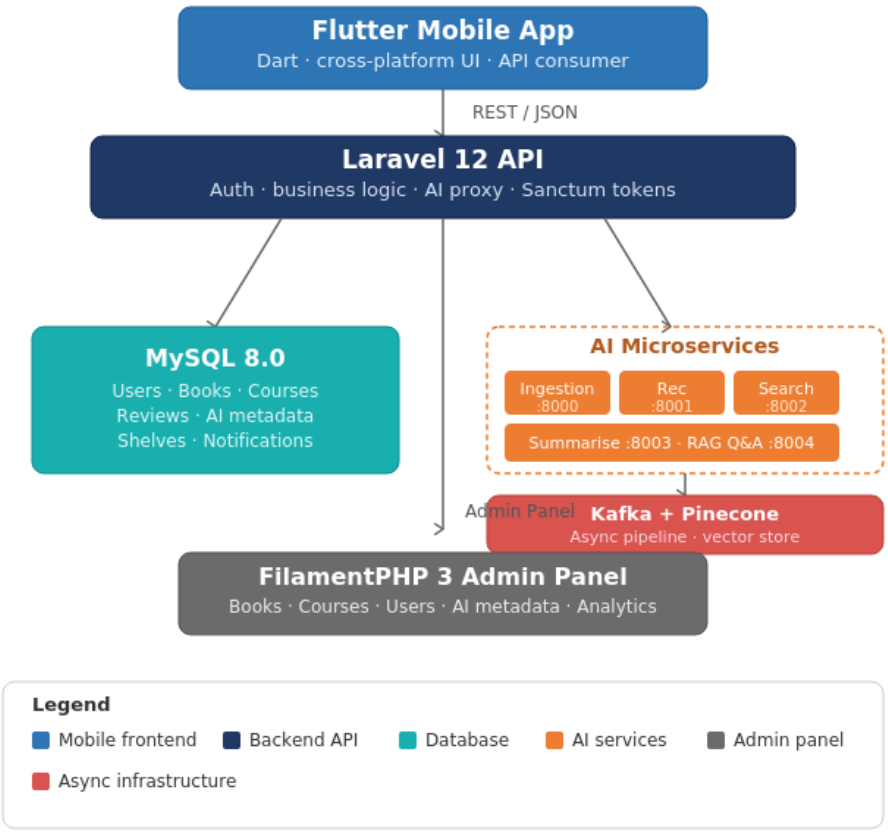


Figure 3.4. Smart AI Library System Architecture

Table 3.1. System Architecture Layers and Responsibilities

Layer	**Technology**	**Responsibility**
Mobile Frontend	Flutter (Dart)	Cross-platform UI, state management, API consumption
Backend API	Laravel 12 (PHP 8.1+)	RESTful endpoints, business logic, authentication, AI proxy
Admin Panel	FilamentPHP 3	Admin UI, CRUD management, analytics dashboard
Database	MySQL 8.0	Persistent storage for users, books, courses, reviews
AI — Ingestion	Python (Port 8000)	PDF/EPUB upload, text extraction, chunking, Kafka
AI — Embedding	Python (Background)	Kafka consumer, vector generation, Pinecone storage
AI — Recommendations	Python (Port 8001)	NCF + CBF hybrid recommendation engine
AI — Semantic Search	Python (Port 8002)	Semantic search and result reranking
AI — Summarization	Python (Port 8003)	FLAN-T5 book summarization
AI — RAG Q&A	Python (Port 8004)	Groq LLM Q&A with citations (sync + SSE streaming)

3.4.1 AI Proxy Architecture

All AI-related API calls follow a proxy pattern. The mobile app sends requests to Laravel, which forwards them to the appropriate AI microservice and returns the result. If an AI service is unavailable, Laravel returns a graceful error message rather than crashing the application. Figure 3.5 illustrates this flow across all six microservices, with every AI request authenticated via Sanctum, validated, then forwarded to the appropriate service.

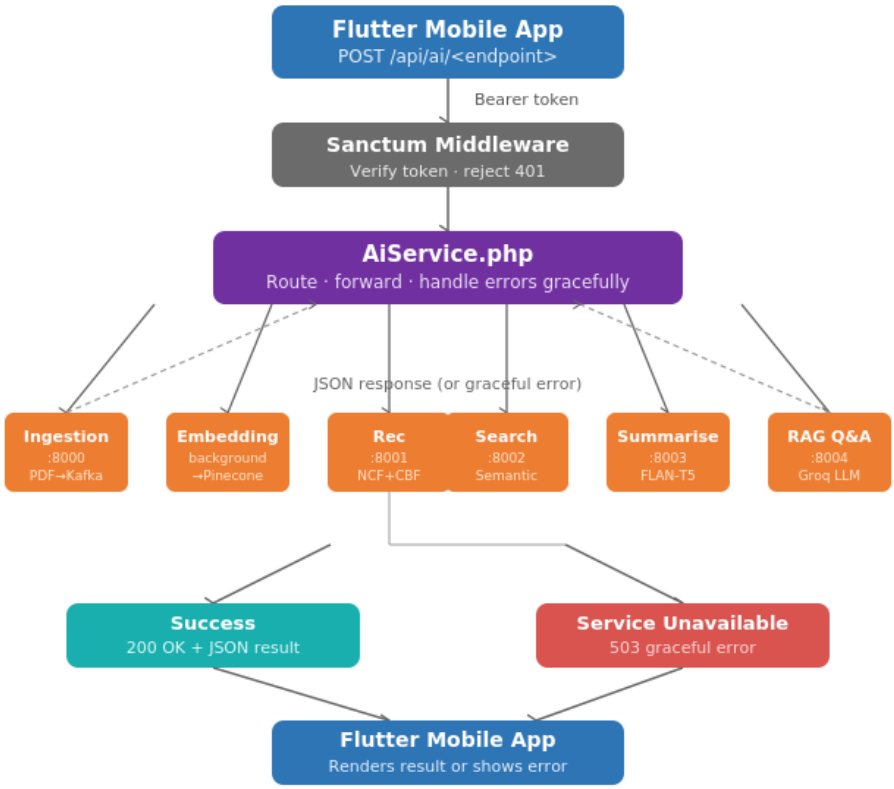


Figure 3.5. AI Proxy Request Routing Across Six Microservices

3.5 Database Design

The system uses **MySQL 8.0** as the primary relational database. The schema is normalized with clear foreign-key relationships between all entities. Laravel 12 migrations are used to version-control the schema. The database covers user management, book and course content, AI metadata, community features, and personal library management. Figure 3.6 summarizes the main entities in the database, their key columns, and the relationships connecting them.

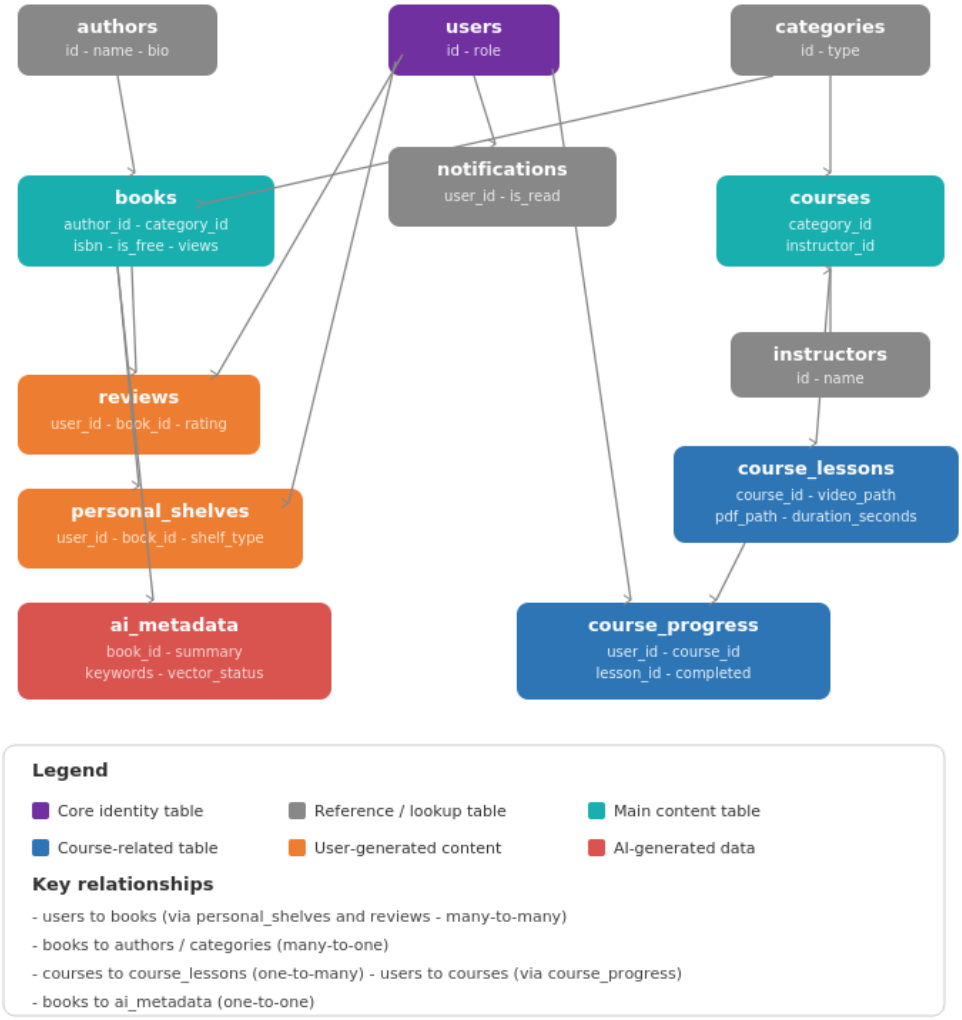


Figure 3.6. Entity Relationship Diagram of the Core Database Tables

3.5.1 users

Table 3.2. users Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
name	VARCHAR(255)	Full name of the user
email	VARCHAR(255)	Unique email address
password	VARCHAR(255)	Bcrypt-hashed password
role	ENUM	Role: admin, researcher, student
email_verified_at	TIMESTAMP	Email verification timestamp
created_at / updated_at	TIMESTAMP	Record timestamps

3.5.2 books

Table 3.3. books Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
title	VARCHAR(255)	Book title
isbn	VARCHAR(255)	Unique ISBN (used as AI service key)
author_id	BIGINT (FK)	Reference to authors table
category_id	BIGINT (FK)	Reference to categories table
description	TEXT	Book synopsis
cover_image	VARCHAR(255)	Path to uploaded cover image
file_path	VARCHAR(255)	Path to PDF/EPUB file for AI ingestion
published_year	YEAR	Year of publication
is_free	BOOLEAN	Whether the book is free or paid
views_count	INT	View counter for trending analytics
created_at / updated_at	TIMESTAMP	Record timestamps

3.5.3 courses

Table 3.4. courses Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
title	VARCHAR(255)	Course title
description	TEXT	Course description
category_id	BIGINT (FK)	Reference to categories table
cover_image	VARCHAR(255)	Course thumbnail image
is_free	BOOLEAN	Whether the course is free or paid
is_featured	BOOLEAN	Flag for featured courses
created_at / updated_at	TIMESTAMP	Record timestamps

3.5.4 course_lessons

Table 3.5. course_lessons Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
course_id	BIGINT (FK)	Reference to courses table
title	VARCHAR(255)	Video lesson title
video_path	VARCHAR(255)	Storage path to uploaded video file (mp4/mov/avi/mkv)
pdf_path	TEXT	Optional storage path to PDF file for the lesson
duration_seconds	INT	Video duration in seconds (default 0)
order	INT	Display order within the course
created_at / updated_at	TIMESTAMP	Record timestamps

3.5.5 reviews

Table 3.6. reviews Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
user_id	BIGINT (FK)	Reference to users table
book_id	BIGINT (FK)	Reference to books table
rating	TINYINT (1–5)	Star rating
comment	TEXT	Optional review text
created_at / updated_at	TIMESTAMP	Record timestamps

3.5.6 bookmarks and personal_shelves

Table 3.7. bookmarks / personal_shelves Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
user_id	BIGINT (FK)	Reference to users table
book_id	BIGINT (FK)	Reference to books table
shelf_type	ENUM	Type: reading, completed, favorite
created_at / updated_at	TIMESTAMP	Record timestamps

3.5.7 ai_metadata

Table 3.8. ai_metadata Table Schema

Column	Type	Description
id	BIGINT (PK)	Auto-incremented primary key
book_id	BIGINT (FK)	Reference to books table
summary	TEXT	AI-generated summary (FLAN-T5)
keywords	JSON	Extracted keywords array
vector_status	ENUM	Embedding status: pending, processing, done
mindmap_data	JSON	Optional visual mind map data
created_at / updated_at	TIMESTAMP	Record timestamps

3.6 API Documentation

The backend exposes a comprehensive RESTful API. All endpoints return JSON responses. Authentication is handled via Laravel Sanctum Bearer tokens passed in the `Authorization` header. The base URL for all API calls is `http://localhost:8000/api`. Figure 3.7 illustrates two complete request lifecycles: a semantic search request proxied to an AI microservice, and a multipart course lesson upload handled directly by Laravel.

Figure 3.7. API Request and Response Lifecycle for Two Representative Endpoints

3.6.1 Authentication Endpoints

Table 3.9. Authentication API Endpoints

Method	Endpoint	Description
POST	/api/register	Register a new user account
POST	/api/login	Login and receive Sanctum token
POST	/api/logout	Invalidate current token
POST	/api/forgot-password/send-otp	Send OTP for password reset
POST	/api/forgot-password/verify-otp	Verify the received OTP code
POST	/api/forgot-password/reset	Reset password after OTP verification
GET	/api/auth/{provider}/redirect	Redirect to Google or Facebook OAuth
GET	/api/auth/{provider}/callback	OAuth callback handler (social login)

3.6.2 Books and Discovery Endpoints

Table 3.10. Books and Discovery API Endpoints

Method	Endpoint	Description
GET	/api/home	Home page: personalized recommendations
GET	/api/discover	Discover page: trending and featured content
GET	/api/discover/trending-20	Top 20 trending books by views count
GET	/api/discover/free-20	Top 20 free books
GET	/api/search	General search across books and content
GET	/api/search/global	Global comprehensive search
GET	/api/search/filter/{type}	Filtered search by type (book/author/course)
DELETE	/api/search/history/{id?}	Delete search history entry
POST	/api/books/{id}/favorite	Toggle book favorite status
POST	/api/books/{id}/listen	Record a book listen/view event
GET	/api/books/{id}/share	Get shareable link for a book

3.6.3 Courses Endpoints

Table 3.11. Courses API Endpoints

Method	Endpoint	Description
GET	/api/courses	List all available courses (no pagination)
GET	/api/courses/home	Course home page (featured + recommended)
GET	/api/courses/discover	Discover page: trending and filtered courses
GET	/api/courses/search	Search courses by title or keyword
GET	/api/courses/{id}	Get course details with lessons and instructor
POST	/api/courses	Create a new course with optional thumbnail upload
PUT	/api/courses/{id}	Update progress (lesson_id, position, completed)
DELETE	/api/courses/{id}	Delete a course (Admin only)
GET	/api/courses/{id}/videos	Get all lessons for a course (with video + PDF URLs)
POST	/api/courses/{id}/videos	Upload new lesson (video + optional PDF, multipart)
PUT	/api/courses/continue-watching	Get courses the user started but not yet completed
DELETE	/api/courses/{courseId}/ lessons/{lessonId}	Delete lesson and remove its files from storage

Figure 3.8 shows the complete flow for the courses module, separated into the student-facing flow (browsing, watching, tracking progress) and the admin/instructor flow (creating courses and uploading lessons).

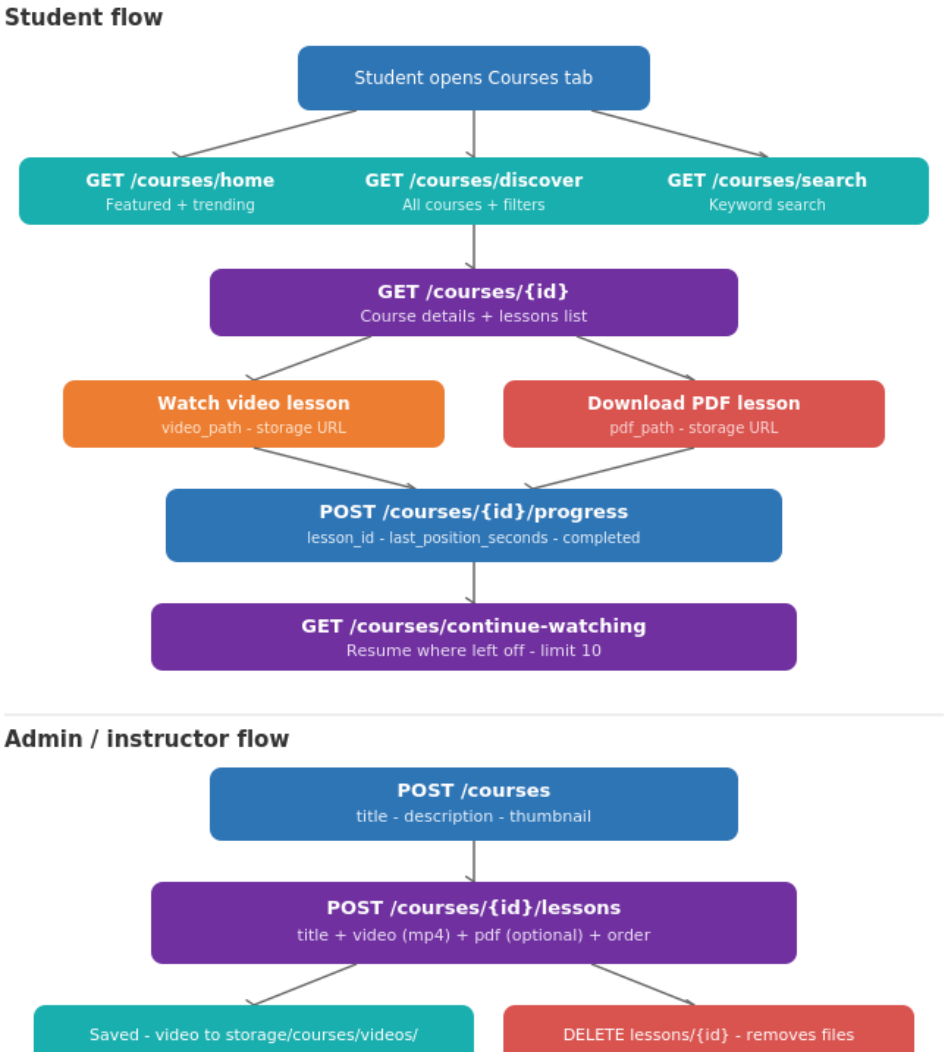


Figure 3.8. Courses Module Flow for Students and Instructors

3.6.4 Favorites and Authors Endpoints

Table 3.12. Favorites and Authors API Endpoints

Method	Endpoint	Description
GET	/api/favorites/{type}	List favorites by type: book, author, narrator
POST	/api/favorites/book/{id}	Toggle book favorite
POST	/api/favorites/author/{id}	Toggle author follow
GET	/api/authors	List all authors
GET	/api/authors/{id}/profile	Get author profile and their books
GET	/api/authors/{id}/similar	Get similar authors

3.6.5 Onboarding and Profile Endpoints

Table 3.13. Onboarding and Profile API Endpoints

Method	Endpoint	Description
POST	/api/profile/update	Update user profile information
GET	/api/topics	Get list of available interest topics
POST	/api/topics/save	Save user topic preferences for personalization

3.6.6 AI Services Endpoints

All AI endpoints follow a proxy pattern: Laravel receives the request, validates authentication, and forwards to the appropriate Python microservice. If a service is unavailable, a graceful error is returned.

Table 3.14. AI Services API Endpoints

Method	Endpoint	Service	Description
GET	/api/ai/health	All	Health check for all 5 AI microservices
POST	/api/ai/ingest	:8000	Upload PDF/EPUB for processing (multipart: file, isbn, title, author)
GET	/api/ai/ingest/{isbn}/status	:8000	Track ingestion pipeline progress
POST	/api/ai/recommendations	:8001	Get personalized book recommendations {user_id, top_k}
GET	/api/ai/recommendations/cold-start	:8001	Cold-start recommendations {genre, limit}
POST	/api/ai/search/semantic	:8002	Semantic search {query, top_k, filter_genres}
POST	/api/ai/search/similar/{isbn}	:8002	Find similar books by ISBN
POST	/api/ai/summarise	:8003	Summarize a book {isbn, summary_type, include_mindmap}
GET	/api/ai/summarise/{isbn}/progress	:8003	Track summarization progress
POST	/api/ai/qa/sync	:8004	Full Q&A response with citations
POST	/api/ai/qa/stream	:8004	Streaming Q&A via SSE tokens

3.7 AI Microservices

The AI layer consists of 6 Python microservices deployed via Docker Compose. Each service is independently scalable and communicates through well-defined REST interfaces. The services use **Kafka** for asynchronous pipeline processing and **Pinecone** as the vector database for semantic search capabilities.

Table 3.15. Python AI Microservices Overview

Service	Port	Model/Tech	Function
ingestion	8000	PyMuPDF + Kafka	PDF/EPUB upload, text extraction, chunking, publish to Kafka
embedding	—	Kafka + Pinecone	Background: generate vectors, store in Pinecone
rec_service	8001	NCF + CBF hybrid	Personalized recommendations based on history and ratings
search_service	8002	Hugging Face + re-ranker	Semantic search with intent understanding
summarise_service	8003	FLAN-T5	Multi-type summarization: short, detailed, or visual mind map
rag_service	8004	Groq LLM	Q&A from book text with citations (sync + SSE)

3.7.1 RAG Q&A Request Sequence

Figure 3.9 traces a single RAG Q&A request end-to-end: a user asks a question about a book through the mobile app, Laravel validates the Sanctum token and proxies the call to the `rag_service`, the AI service retrieves the most relevant chunks from Pinecone and generates a cited answer with the Groq LLM, and the response flows back to the user. The alt frame shows the graceful-error path Laravel takes when the AI microservice is unavailable.

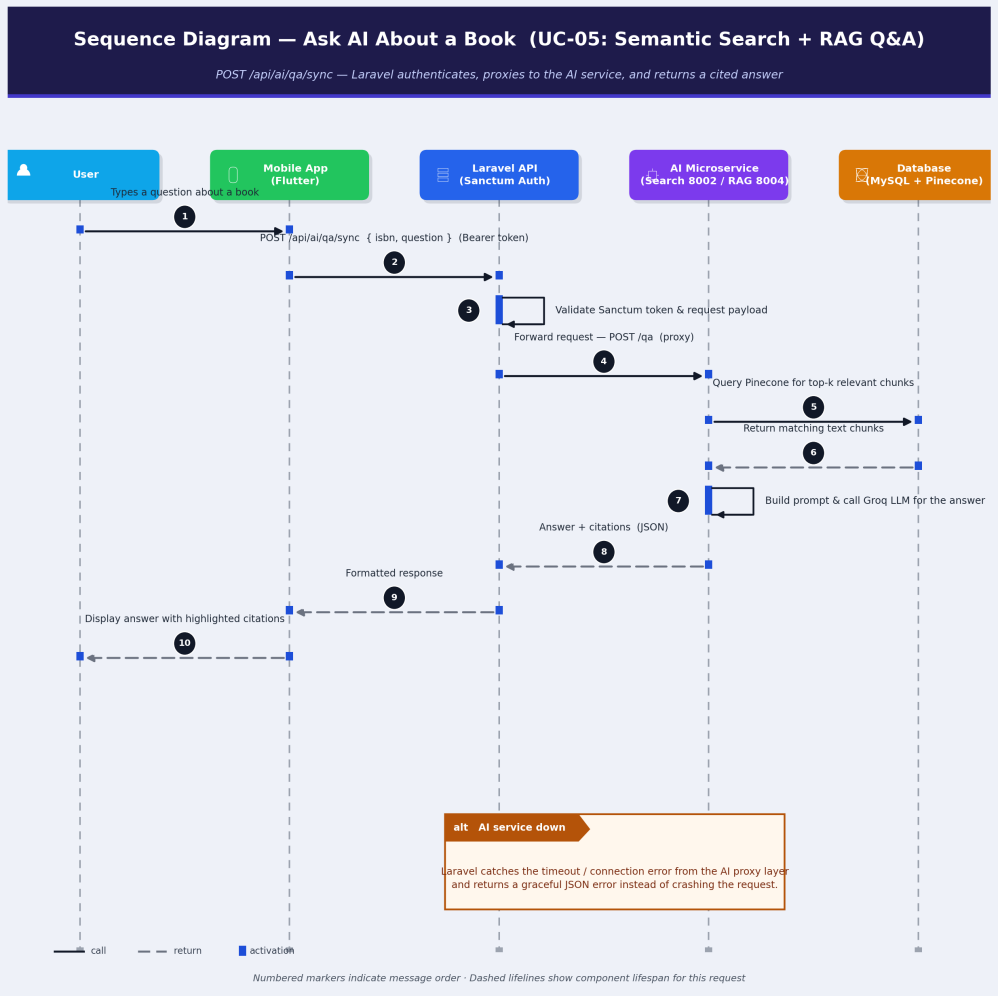

Sequence Diagram — Ask AI About a Book (UC-05: Semantic Search + RAG Q&A)
POST /api/ai/qa/sync — Laravel authenticates, proxies to the AI service, and returns a cited answer
User
Mobile App (Flutter)
Laravel API (Sanctum Auth)
AI Microservice (Search 8002 / RAG 8004)
Database (MySQL + Pinecone)
Types a question about a book
1
POST /api/ai/qa/sync { isbn, question } (Bearer token)
2
3
Validate Sanctum token & request payload
Forward request — POST /qa (proxy)
4
Query Pinecone for top-k relevant chunks
5
Return matching text chunks
6
7
Build prompt & call Groq LLM for the answer
Answer + citations (JSON)
8
Formatted response
9
Display answer with highlighted citations
10
alt AI service down
Laravel catches the timeout / connection error from the AI proxy layer and returns a graceful JSON error instead of crashing the request.
call
return
activation
Numbered markers indicate message order · Dashed lifelines show component lifespan for this request

Figure 3.9. Sequence Diagram for the AI-Powered Semantic Search and Q&A Request

3.7.2 Courses AI — TranscribeAllLessons

As part of the courses AI integration, a dedicated Laravel Artisan command `TranscribeAllLessons` was developed. This command processes course video lessons by sending them to the AI transcription pipeline, generating text transcripts stored in the `course_videos` table. This enables searchable lesson content, AI-powered lesson summaries, and accessibility improvements for hearing-impaired users.

Listing 3.1. Running the TranscribeAllLessons command

```
# Run transcription for all unprocessed lessons
php artisan transcribe:all-lessons

# The command:
# 1. Fetches all course_videos without transcripts
# 2. Sends each video to the AI transcription service
# 3. Stores the returned transcript in the database
# 4. Logs progress and handles failures gracefully
```

3.7.3 AI Service Configuration

All AI service URLs and timeout settings are centralized in `config/ai.php`. The `AiService.php` class in `app/Services` provides a unified interface for all AI microservice calls, handling connection errors gracefully.

Table 3.16. AI Service Configuration Parameters

Config Key	Default Value	Description
ai.ingestion_url	http://localhost:8000	Ingestion service base URL
ai.rec_url	http://localhost:8001	Recommendation service base URL
ai.search_url	http://localhost:8002	Semantic search service base URL
ai.summarise_url	http://localhost:8003	Summarization service base URL
ai.rag_url	http://localhost:8004	RAG Q&A service base URL
ai.timeout	30 seconds	Default HTTP timeout for AI requests

3.8 Admin Panel (FilamentPHP 3)

The backend includes a full-featured admin panel built with **FilamentPHP 3**, accessible at `/admin`. The panel is organized into four sections covering the entire platform, as shown in Figure 3.10.

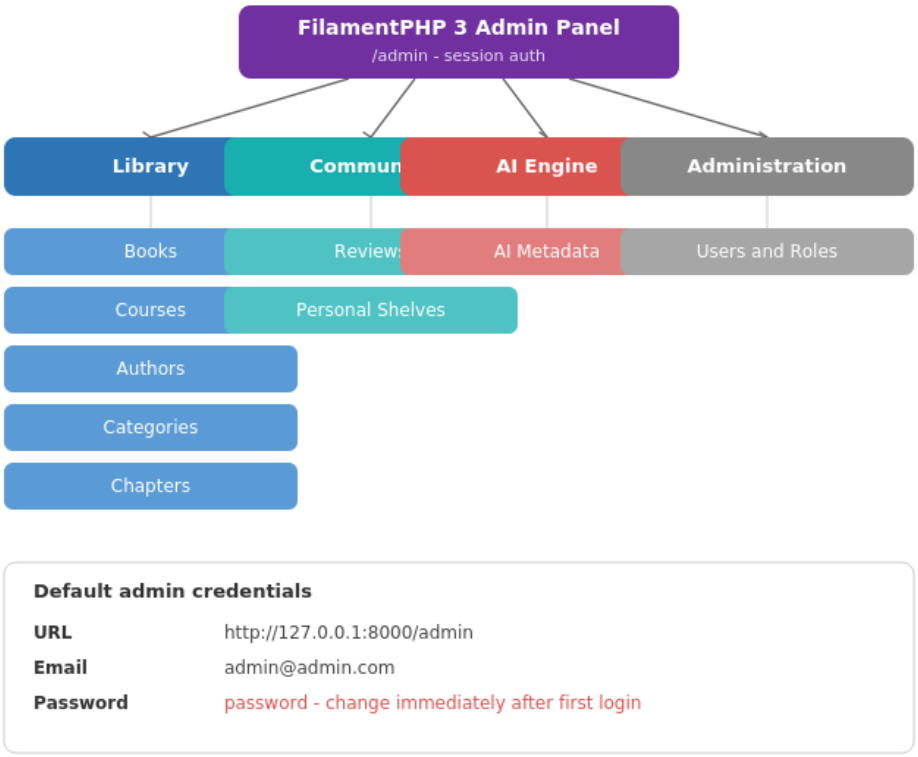


Figure 3.10. FilamentPHP Admin Panel Sections and Resources

Table 3.17. Admin Panel Sections and Resources

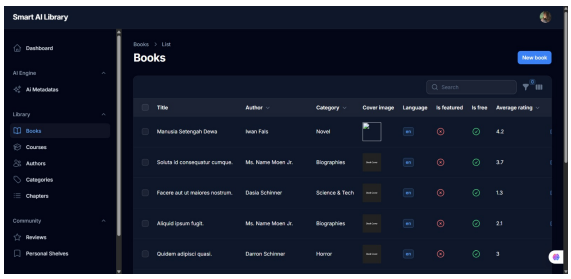
Section	Resources	Description
Library	Books, Courses, Authors, Categories, Instructors, Chapters	Full CRUD with file uploads
Community	Reviews, Personal Shelves	User-generated content moderation
AI Engine	AI Metadata	View summaries, keywords, vector status
Administration	Users	User management with role assignment

Table 3.18. Admin Panel Default Credentials

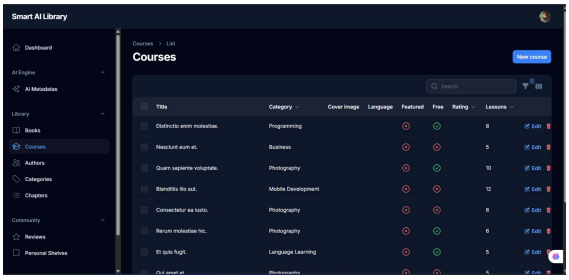
Field	Value
Admin Panel URL	http://127.0.0.1:8000/admin
Default Email	admin@admin.com
Default Password	password
Note	Change credentials immediately after first deployment

3.8.1 Library Resource Screens

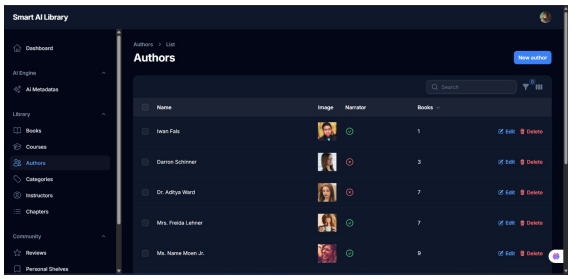
Figure 3.11 shows the six Library-section resource list views, each following the same FilamentPHP table pattern: a searchable, sortable data table with inline Edit/Delete actions and a primary "New {resource}" action button in the top-right corner.



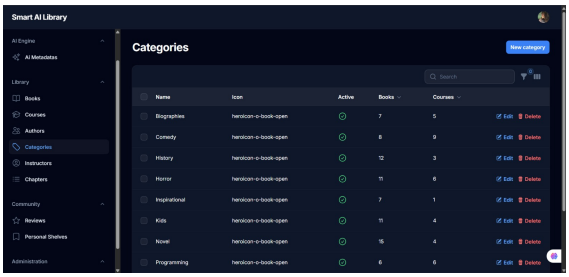
(a) Books



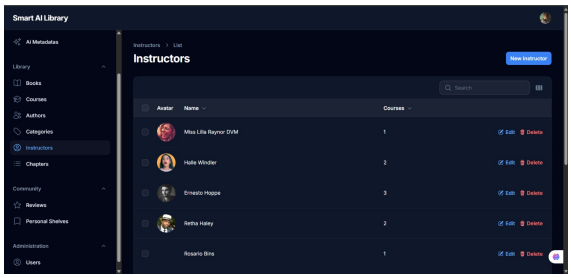
(b) Courses



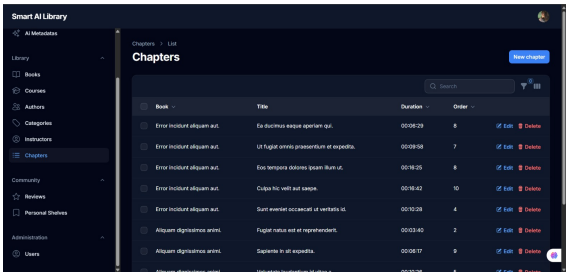
(c) Authors



(d) Categories



(e) Instructors

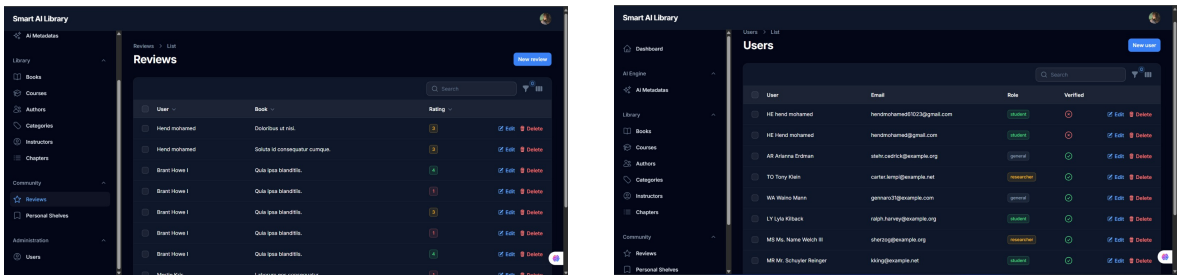


(f) Chapters

Figure 3.11. Library Section — Resource List Views in the Admin Panel

3.8.2 Community and Administration Screens

The Community section moderates user-generated content, while the Administration section manages platform users and their assigned roles, as shown in Figure 3.12.



(a) Reviews — Community Section **(b)** Users — Administration Section

Figure 3.12. Community Moderation and User Administration Screens

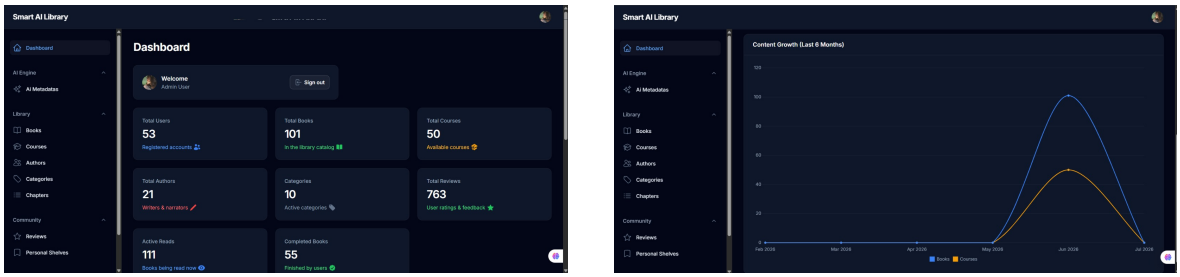
3.8.3 Dashboard Analytics

The admin panel's main Dashboard page gives administrators an at-a-glance overview of platform activity by aggregating live counts and trends directly from the MySQL database. Each statistic widget is backed by a Filament Stats Overview widget that executes an Eloquent aggregate query whenever the dashboard is loaded. Therefore, all displayed values always reflect the latest state of the database without requiring manual refresh.

Table 3.19. Dashboard Statistics

Statistic	Source Model	Description
Total Users	User	Total registered users across all roles.
Total Books	Book	Total books currently stored in the library.
Total Courses	Course	Number of available courses.
Total Authors	Author	Total registered authors.
Categories	Category	Total available categories.
Total Reviews	Review	Number of user reviews and ratings.
Active Reads	PersonalShelf	Books currently marked as reading.
Completed Books	PersonalShelf	Books marked as completed by users.

The dashboard also includes a Content Growth chart that visualizes the number of books and courses added during the previous six months. The chart is generated using Filament Chart Widgets and groups records according to their creation month, allowing administrators to monitor content growth trends over time. Figure 3.13 shows the live dashboard: the statistic cards described in Table **??** on the left, and the resulting Content Growth chart on the right.



(a) Dashboard Statistic Cards **(b)** Content Growth Chart (Last 6 Months)

Figure 3.13. Live Admin Dashboard — Statistics Overview and Content Growth Trend

3.9 Testing

The backend APIs were tested using **Postman** for manual endpoint validation and Laravel's built-in **PHPUnit** framework for automated tests. Each endpoint was verified for correct HTTP response codes, data accuracy, authentication enforcement, and error handling for invalid inputs.

3.9.1 Testing Tools

Table 3.20. Testing Tools and Scope

Tool	Scope
Postman	Manual API testing: auth flows, CRUD operations, AI endpoints, edge cases
PHPUnit	Automated unit and integration tests for models and API responses
Browser (Manual)	Admin panel CRUD, file uploads, dashboard analytics
Docker Health Checks	Verify all 5 AI microservices via `/health` endpoints

3.9.2 Sample Test Cases

Table 3.21. Sample API Test Cases and Expected Results

Endpoint	Method	Test Case	Expected Result
/api/register	POST	Valid registration data	201 Created + User object
/api/register	POST	Duplicate email	422 Unprocessable Entity
/api/login	POST	Valid credentials	200 OK + Sanctum token
/api/login	POST	Wrong password	401 Unauthorized
/api/forgot-password/ send-otp	POST	Valid phone number	200 OK + OTP sent
/api/forgot-password/ verify-otp	POST	Wrong OTP code	422 Validation Error
/api/discover/trending-20	GET	Fetch trending books	200 OK + 20 books array
/api/courses	GET	List all courses	200 OK + Paginated courses
/api/courses/{id}/ videos	GET	Valid course ID	200 OK + Videos array
/api/courses/{id}/ videos	POST	Create video (Admin)	201 Created + Video
/api/ai/health	GET	All services running	200 OK + All services healthy
/api/ai/search/ semantic	POST	Valid semantic query	200 OK + Ranked results
/api/ai/ recommendations	POST	Valid user_id	200 OK + Recommended books
/api/ai/summarise	POST	Valid ISBN + type	200 OK + Summary text
/api/ai/qa/sync	POST	Question about book	200 OK + Answer + citations
/api/favorites/ book/{id}	POST	Toggle favorite (auth)	200 OK + Status updated
/api/bookmarks	GET	No auth token	401 Unauthorized

3.10 Challenges Faced

- **AI Microservices Integration:** Designing a proxy architecture in Laravel that gracefully handles failures from any of the six Python AI microservices required careful timeout handling, retry mechanisms, and centralized error management through the AiService class.
- **Docker Environment Configuration:** Setting up Kafka, Pinecone, and the containerized AI services required configuring environment variables such as `PINECONE_API_KEY`, `GROQ_API_KEY`, and `OPENAI_API_KEY`, while ensuring reliable communication between containers.
- **Courses AI Integration:** Implementing the TranscribeAllLessons Artisan command required handling large video files, retry logic, and batch processing while keeping the application responsive.
- **Dual Authentication System:** Combining Laravel Sanctum token authentication with FilamentPHP session authentication required custom guard configuration and middleware separation.
- **Social Login Integration:** Supporting Google and Facebook authentication required configuring Laravel Socialite, callback handling, and account linking for existing users.
- **Role-Based Access Control:** Protecting administration endpoints while allowing public access to discovery features required extensive permission testing across different user roles.
- **Database Migration Ordering:** Managing foreign-key dependencies across related tables required careful migration ordering to avoid integrity constraint violations.

3.11 Future Work

- Merge the Courses AI branch (TranscribeAllLessons) into the main branch and extend it to support real-time transcription during video upload.
- Add AI-generated audio books using modern Text-to-Speech models.
- Implement OCR support for scanned books using Tesseract or similar technologies.
- Develop advanced dashboard analytics including user engagement, AI usage, and reading statistics.
- Extend the recommendation engine to recommend courses alongside books.

- Add multilingual support (Arabic and English) across all APIs and the admin panel.
- Implement offline reading with local caching and synchronization.
- Integrate secure payment processing for premium content.
- Enable RAG-based Question Answering directly on course lesson transcripts.

Chapter Summary

This chapter documented the backend implementation of Learnova — a robust, secure, and AI-integrated system built on Laravel 12, FilamentPHP 3, MySQL 8.0, and a suite of 6 Python AI microservices. The proxy architecture ensures that the platform degrades gracefully when AI services are unavailable, maintaining core functionality at all times. The `TranscribeAllLessons` command demonstrates the system's extensibility, enabling AI features to be added incrementally to existing content modules. The admin panel provides administrators with a powerful graphical interface for managing all platform content. The next chapter details the intelligent AI features built on top of this backend infrastructure.

Chapter 4

Artificial Intelligence Integration

Contents

4.1 Ingestion Service — Document Processing Pipeline 48
4.2 Embedding Service — Semantic Vectorisation 50
4.3 Search Service — Two-Stage Semantic Retrieval 52
4.4 Summarisation Service — FLAN-T5 Map-Reduce & Mind Maps 54
4.5 RAG Service — Retrieval-Augmented Generation Q&A 57
4.6 Recommendation Service — Hybrid NCF + CBF 60
4.7 Mobile Client Integration — Lesson Assistant and Lightweight Fallback Pathway 62
4.8 Ethical Considerations 65
4.9 Chapter Summary 66

This chapter provides a comprehensive technical description of the AI subsystems embedded in the AIGP Smart Library platform. The primary pipeline comprises six services — the Ingestion Service, the Embedding Service, the Search Service, the Summarisation Service, the Retrieval-Augmented Generation (RAG) Q&A Service, and the Recommendation Service — built around a persistent Pinecone vector index and Kafka-driven event processing. Together, these services transform a static collection of book files into a dynamic, queryable knowledge base that users can explore through natural language, receive personalised recommendations, and query for specific information — without any manual indexing or metadata entry. Section 4.7 then describes a complementary, lightweight AI pathway implemented directly in the mobile client's Laravel backend, which extends AI assistance to course-lesson video content and provides a fallback for books that fall outside the vector-indexed catalogue.

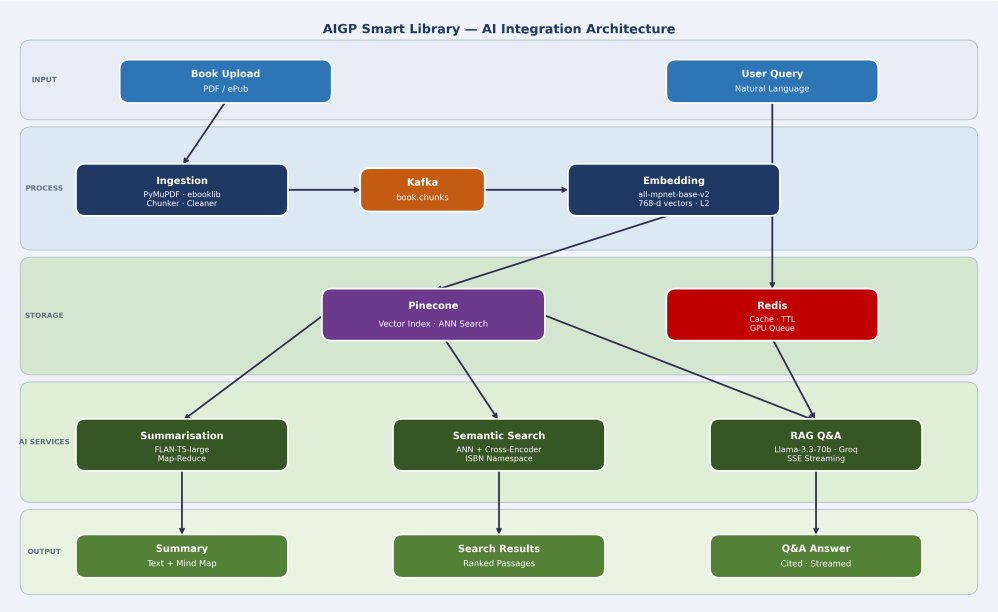
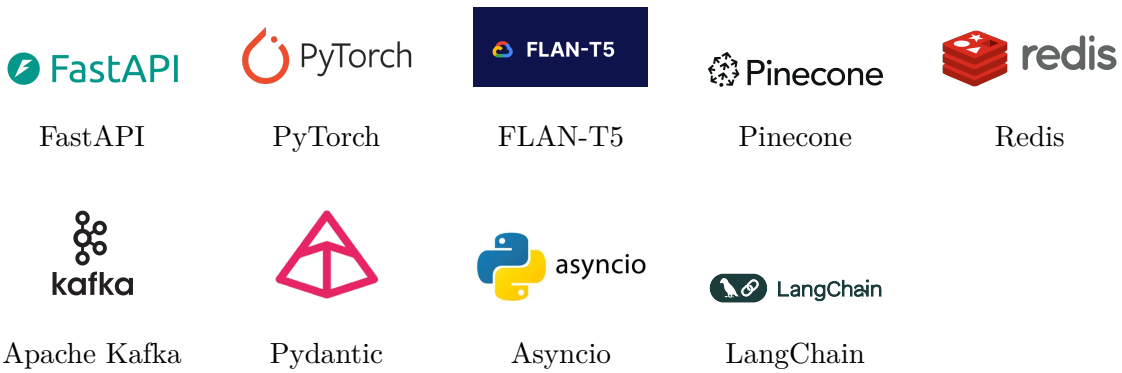


Figure 4.1. Overview of the AIGP Smart Library AI Integration Architecture



FastAPI PyTorch FLAN-T5 Pinecone Redis

Apache Kafka Pydantic Asyncio LangChain

Figure 4.2. Technology Stack Used in the AI Integration Layer

4.1 Ingestion Service — Document Processing Pipeline

The Ingestion Service is the entry point for all book content in the platform. It accepts PDF and ePub files uploaded by library administrators and orchestrates a three-stage pipeline: text extraction, cleaning and sentence tokenisation, and sliding-window chunking. The output is a stream of text chunks published to Kafka, which the Embedding Service then consumes asynchronously to produce vector representations.

4.1.1 Text Extraction

Format detection is based on file extension. PDF files are processed using **PyMuPDF** (`fitz`), which extracts plain text page by page. ePub files are processed using **ebooklib**, which iterates over HTML chapter items, strips all HTML tags, and yields plain text sections. In both cases, pages or sections with fewer than 50 characters are silently skipped, filtering out blank pages, cover images, and decorative separators that would otherwise produce noise in the vector index.

4.1.2 Cleaning and Chunking

Extracted text is cleaned and sentence-tokenised before chunking. The chunker uses the `all-mpnet-base-v2` tokenizer to measure token counts precisely, ensuring that chunks match the embedding model's input space. A sliding-window algorithm fills each chunk greedily up to 512 tokens, then carries the last 50 tokens into the next chunk as overlap — preserving cross-boundary semantic continuity. Sentences that individually exceed 512 tokens are split into sub-windows before chunking begins.

Each chunk is assigned a deterministic identifier of the form `{isbn}_p{page_num}_c{chunk_index}`, enabling exact traceability from any retrieved vector back to its source page in the original book.

Table 4.1. Ingestion Service Chunking Parameters

Parameter	Value	Config Key	Effect
Chunk Size	512 tokens	CHUNK_SIZE	Maximum tokens per chunk
Chunk Overlap	50 tokens	CHUNK_OVERLAP	Tokens carried into next chunk for semantic continuity
Min Page Length	50 characters	(hardcoded)	Filters blank/decorative pages
Upsert Batch Size	100 vectors	UPSERT_BATCH_SIZE	Vectors flushed per Pinecone upsert call

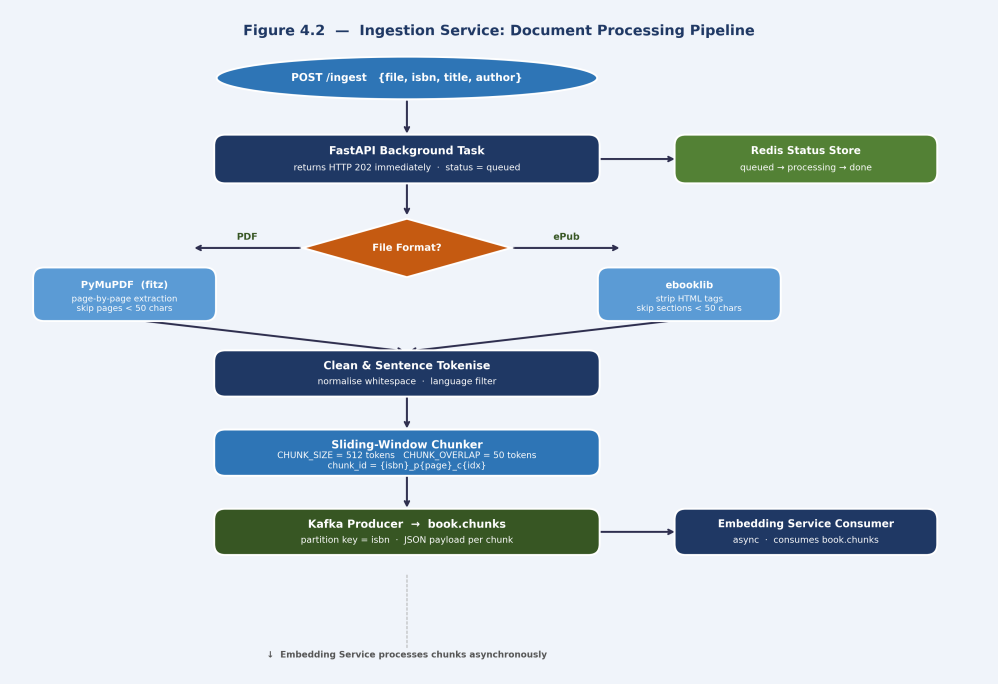


Figure 4.2 — Ingestion Service: Document Processing Pipeline
POST /ingest {file, isbn, title, author}
FastAPI Background Task
returns HTTP 202 immediately · status = queued
Redis Status Store
queued → processing → done
PDF
File Format?
ePub
PyMuPDF (fitz)
page-by-page extraction
skip pages < 50 chars
ebooklib
strip HTML tags
skip sections < 50 chars
Clean & Sentence Tokenise
normalise whitespace · language filter
Sliding-Window Chunker
CHUNK_SIZE = 512 tokens CHUNK_OVERLAP = 50 tokens
chunk_id = {isbn}_p{page}_c{idx}
Kafka Producer → book.chunks
partition key = isbn · JSON payload per chunk
Embedding Service Consumer
async · consumes book.chunks
↓ Embedding Service processes chunks asynchronously

Figure 4.3. Ingestion Service — Document Processing Pipeline

4.1.3 Kafka Publishing and Status Tracking

Each chunk is serialised as a JSON message and published to the Kafka topic `book.chunks` with the book's ISBN as the partition key, ensuring all chunks from the same book are processed in order by the same Embedding Service consumer instance. Ingestion runs as a FastAPI background task: the API accepts the upload, stores the file to disk, and immediately returns HTTP 202 Accepted with a job ID. The caller can poll `/ingest/{isbn}/status` to track progress through the `queued` → `processing` → `done` lifecycle, with status persisted in Redis.

4.2 Embedding Service — Semantic Vectorisation

The Embedding Service is the foundational layer of the platform's AI pipeline. It converts raw text chunks, produced by the Ingestion Service, into dense 768-dimensional numerical vectors that encode semantic meaning. These vectors are stored in Pinecone and serve as the retrieval backbone for every downstream AI capability: semantic search, recommendations, summarisation, and RAG Q&A.

4.2.1 Design Rationale

Traditional library search systems rely on keyword matching over metadata fields such as title, author, and ISBN. This approach fails to surface semantically relevant content buried within the book text itself. Semantic search addresses this limitation by representing every passage as a point in a shared vector space: two passages with similar meaning are geometrically close, even if they share no words in common.

The Embedding Service is designed as a headless Kafka consumer with no HTTP interface of its own. This architectural choice decouples embedding from the upload API, allowing large books to be indexed asynchronously without blocking the user-facing request. Multiple consumer instances can be deployed independently to scale ingestion throughput.

4.2.2 Model: all-mpnet-base-v2

The `sentence-transformers/all-mpnet-base-v2` model was selected as the embedding backbone after evaluation on the Semantic Textual Similarity (STS) benchmark. It is a fine-tuned variant of Microsoft's MPNet architecture, trained on over one billion sentence pairs, producing 768-dimensional vectors with state-of-the-art performance on passage-level retrieval tasks.

A critical mathematical property is L2 normalisation: all output vectors are projected onto the unit hypersphere. Under this normalisation, cosine similarity equals the dot product of two vectors — a property exploited by Pinecone's ANN index through maximum inner-product search, enabling sub-millisecond retrieval across millions of vectors.

Table 4.2. Embedding Model Configuration and Design Decisions

Property	Value	Justification
Model	all-mpnet-base-v2	Top STS benchmark performance on passage retrieval tasks
Output Dimension	768-d float32	Balance between representational power and storage cost
Normalisation	L2 (unit sphere)	Cosine similarity = dot product — exploited by Pinecone ANN
Batch Size	64 texts / forward pass	Optimal GPU utilisation without out-of-memory risk
Hardware	CUDA GPU (CPU fallback)	Automatic device selection at runtime
Lazy Loading	On first encode() call	Avoids GPU initialisation overhead at service startup

4.2.3 BookEncoder Implementation

The `BookEncoder` class wraps the SentenceTransformer library and exposes two methods: `encode_single()` for real-time query encoding and `encode_batch()` for bulk ingestion. Text is pre-processed by the Ingestion Service into sliding-window chunks of 512 tokens with a 50-token overlap before being passed to the encoder. As shown in the code excerpt below, the `torch.no_grad()` context manager disables gradient computation during inference, reducing memory usage and improving throughput by approximately 30%.

Listing 4.1. BookEncoder encode_batch method

```
def encode_batch(self, texts: list[str], batch_size: int = 64) ->
    np.ndarray:
     model = self._ensure_model()        # lazy-load on first call
     with torch.no_grad():               # disable gradients for
        inference
         vecs = model.encode(
             texts, batch_size=batch_size,
             convert_to_numpy=True,
             normalize_embeddings=True, # apply L2 normalisation
         )
     return vecs.astype(np.float32)      # 768-d float32 matrix
```

4.2.4 Kafka-to-Pinecone Pipeline & Fault Tolerance

The `EmbeddingBatchJob` class buffers up to 100 vector records in memory and flushes them to Pinecone as a single batched upsert, reducing API call overhead. Each vector is stored in a Pinecone namespace equal to the book's ISBN, enabling efficient per-book filtering without maintaining separate indexes.

Upsert operations are retried up to three times with exponential back-off (delays of 1s, 2s, and 4s). If all three attempts fail, the batch is forwarded to a Kafka dead-letter topic (`book.chunks.dlq`) for manual inspection and replay. This two-layer failure handling prevents data loss while avoiding infinite retry loops that could stall the consumer pipeline.

Table 4.3. Embedding Batch Job Processing Stages

Stage	**Input**	**Output**	**Technology**
Consume	Kafka: book.chunks	ChunkData object	aiokafka KafkaConsumer
Encode	Chunk text (string)	768-d float32 vector	SentenceTransformer encode_single()
Buffer	Vector record + metadata	In-memory list (max 100)	Python list
Upsert	List of records	Pinecone index entry	Pinecone client upsert()
DLQ	Failed batch	Kafka: book.chunks.dlq	aiokafka KafkaProducer

4.3 Search Service — Two-Stage Semantic Retrieval

The Search Service provides natural-language book search across the entire library catalogue. Unlike traditional keyword search, the service encodes the query into a 768-dimensional vector and retrieves semantically similar passages — surfacing relevant content even when the query shares no exact words with the text. To balance recall and precision, the service implements a two-stage retrieval pipeline combining a fast bi-encoder for candidate recall with a slower but more accurate cross-encoder for re-ranking.

4.3.1 Two-Stage Pipeline

Stage 1 — Approximate Nearest Neighbour (ANN) Retrieval: The user's query is encoded by the same `all-mpnet-base-v2` bi-encoder used during ingestion, producing an L2-normalised 768-dimensional vector. This vector is submitted to Pinecone's global namespace, which contains all chunk vectors across all books, retrieving the top 50 candidates (`ANN_TOP_K = 50`) by cosine similarity. An optional genre filter can be applied as a Pinecone metadata filter to restrict results to specific categories.

Stage 2 — Cross-Encoder Re-ranking: The 50 ANN candidates are passed to a CrossEncoder model (`cross-encoder/ms-marco-MiniLM-L-6-v2`), which scores each query–passage pair jointly. Unlike the bi-encoder, which encodes query and passage independently, the cross-encoder attends to both simultaneously — producing more accurate relevance scores at the cost of higher latency. The top 10 results (`FINAL_TOP_K = 10`) by cross-encoder score are returned to the caller.

Table 4.4. Search Service Two-Stage Pipeline

Stage	**Model**	**Output**
ANN Retrieval	all-mpnet-base-v2 (bi-encoder)	Top 50 candidates from Pinecone global namespace
Re-ranking	cross-encoder/ms-marco-MiniLM-L-6-v2	Top 10 results sorted by cross-encoder score
Similar Books	CBF (Pinecone synopsis namespace)	Books semantically similar to a given ISBN

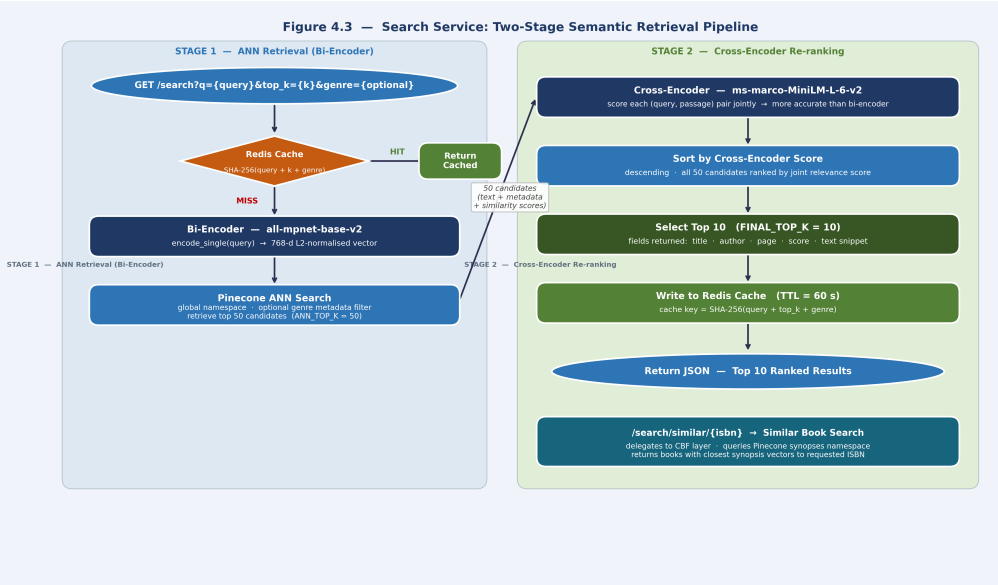


Figure 4.3 — Search Service: Two-Stage Semantic Retrieval Pipeline
STAGE 1 — ANN Retrieval (Bi-Encoder)
GET /search?q={query}&top_k={k}&genre={optional}
Redis Cache
SHA-256(query + k + genre)
HIT
Return Cached
MISS
Bi-Encoder — all-mpnet-base-v2
encode_single(query) → 768-d L2-normalised vector
Pinecone ANN Search
global namespace · optional genre metadata filter
retrieve top 50 candidates (ANN_TOP_K = 50)
50 candidates
(text + metadata
+ similarity scores)
STAGE 2 — Cross-Encoder Re-ranking
Cross-Encoder — ms-marco-MiniLM-L-6-v2
score each (query, passage) pair jointly → more accurate than bi-encoder
Sort by Cross-Encoder Score
descending · all 50 candidates ranked by joint relevance score
Select Top 10 (FINAL_TOP_K = 10)
fields returned: title · author · page · score · text snippet
Write to Redis Cache (TTL = 60 s)
cache key = SHA-256(query + top_k + genre)
Return JSON — Top 10 Ranked Results
/search/similar/{isbn} → Similar Book Search
delegates to CBF layer · queries Pinecone synopses namespace
returns books with closest synopsis vectors to requested ISBN

Figure 4.4. Search Service — Two-Stage Semantic Retrieval Pipeline

4.3.2 Caching and Similar-Book Search

Search results are cached in Redis for 60 seconds (`REDIS_TTL_SEARCH`) using a SHA-256 hash of the query, top-k, and genre filter as the cache key. This avoids redundant Pinecone queries for repeated searches within the same minute. The service also exposes a `/search/similar/{isbn}` endpoint that delegates to the Content-Based Filtering layer of the Recommendation Service, returning books whose synopsis vectors are closest to the requested ISBN in Pinecone's synopses namespace.

4.4 Summarisation Service — FLAN-T5 Map-Reduce & Mind Maps

The Summarisation Service generates concise textual summaries of full book content and structured mind-map representations of key themes. Because a full book may contain thousands of text chunks — far exceeding any language model context window — the service employs a map-reduce strategy to handle arbitrarily long documents, as illustrated in Figure 4.5.

4.4.1 Model: FLAN-T5-large

The service uses Google's **FLAN-T5-large**, a sequence-to-sequence model based on the T5 architecture. FLAN-T5 was instruction-fine-tuned on a large collection of tasks expressed as natural language instructions, enabling it to follow structured prompts reliably without task-specific fine-tuning — making it well-suited for a library system where content varies widely in genre and domain.

Beam search (`num_beams=4`) explores multiple candidate sequences simultaneously, consistently producing higher-quality summaries than greedy decoding. A `no_repeat_ngram_size` of 3 suppresses repetitive phrase patterns, a common failure mode in extractive-style models. Inference runs on a dedicated GPU with automatic CPU fallback.

Table 4.5. FLAN-T5 Prompt Configurations by Summary Type

Summary Type	**System Prompt Role**	**Max Output Tokens**	**Use Case**
SHORT	Book blurb writer — 2–3 engaging sentences for a general reader	150	Catalogue listing, quick overview
DETAILED	Literary analyst — thematic summary with key events and characters	300	Book report, deep-dive analysis

4.4.2 Map-Reduce Pipeline

The `MapReducePipeline` processes books in two phases to overcome the context window constraint:

- **MAP Phase:** Chunks are fetched from Pinecone in page-number order and divided into batches of 5 consecutive chunks. Each batch is independently summarised by FLAN-T5 with an 80-token output cap, producing a list of intermediate summaries. Batches are processed sequentially to respect GPU memory constraints.
- **REDUCE Phase:** All intermediate summaries are concatenated into a single string and passed through FLAN-T5 one final time with the user-selected type — SHORT (150 tokens) or DETAILED (300 tokens) — producing the final book summary.

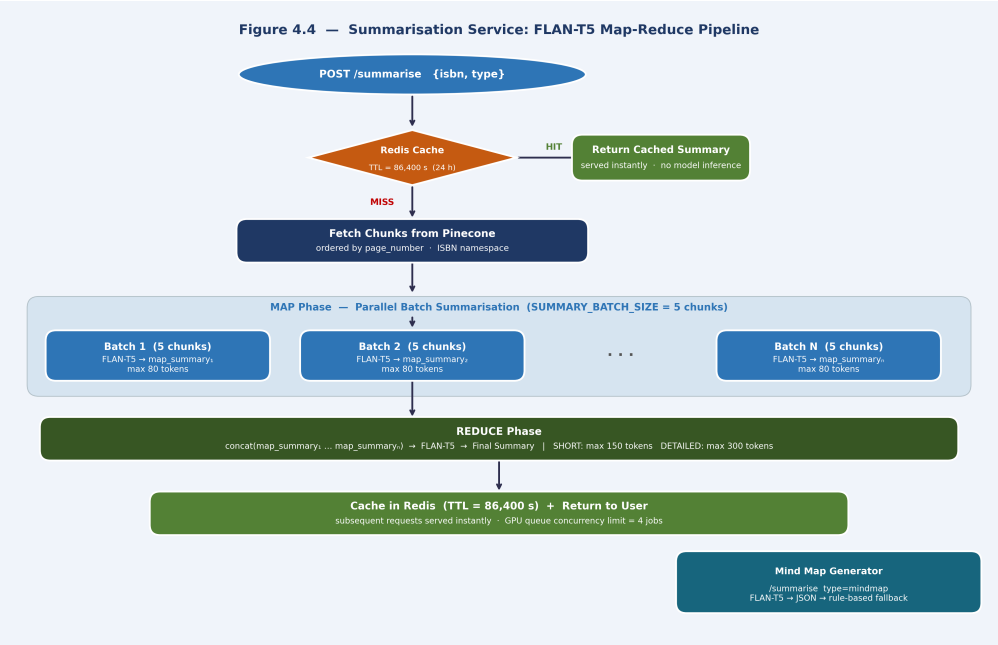


Figure 4.4 — Summarisation Service: FLAN-T5 Map-Reduce Pipeline
POST /summarise {isbn, type}
Redis Cache
TTL = 86,400 s (24 h)
HIT
Return Cached Summary
served instantly · no model inference
MISS
Fetch Chunks from Pinecone
ordered by page_number · ISBN namespace
MAP Phase — Parallel Batch Summarisation (SUMMARY_BATCH_SIZE = 5 chunks)
Batch 1 (5 chunks)
FLAN-T5 → map_summary₁
max 80 tokens
Batch 2 (5 chunks)
FLAN-T5 → map_summary₂
max 80 tokens
...
Batch N (5 chunks)
FLAN-T5 → map_summaryₙ
max 80 tokens
REDUCE Phase
concat(map_summary₁ ... map_summaryₙ) → FLAN-T5 → Final Summary | SHORT: max 150 tokens DETAILED: max 300 tokens
Cache in Redis (TTL = 86,400 s) + Return to User
subsequent requests served instantly · GPU queue concurrency limit = 4 jobs
Mind Map Generator
/summarise type=mindmap
FLAN-T5 → JSON → rule-based fallback

Figure 4.5. Map-Reduce Summarisation Pipeline with Redis Cache and GPU Queue

The final summary is cached in Redis with a TTL of 24 hours (86,400 seconds), so repeated requests for the same book are served instantly without re-running the pipeline. A GPU queue counter in Redis limits concurrent summarisation jobs to a maximum of 4 to prevent out-of-memory errors under simultaneous requests. If the queue is full, the API returns HTTP 503 with a `Retry-After` header.

4.4.3 Mind Map Generation

In addition to textual summaries, the service can generate a structured JSON mind-map tree representing the book's key themes. The `MindMapBuilder` class prompts FLAN-T5 to produce a JSON object with a root node (the book title) and child nodes (main themes), each containing a list of supporting subpoints. All output is validated against a Pydantic schema (`MindMapOutput`). If validation fails due to malformed JSON or missing nodes, a rule-based fallback generator guarantees a valid, well-structured response by grouping summary sentences into child nodes.

Table 4.6. Mind Map Generation Pipeline with Fallback Strategy

Stage	Input	Output	Fallback?
Prompt	Book summary (max 3000 chars)	Structured prompt for FLAN-T5	No
Generation	Prompt + FLAN-T5	Raw text output (may contain JSON)	No
JSON Extraction	Raw text	Parsed dict object	Yes — rule-based
Pydantic Validation	dict object	MindMapOutput (validated)	Yes — rule-based
Response	MindMapOutput	JSON tree: root + children[]	Fallback always valid

4.5 RAG Service — Retrieval-Augmented Generation Q&A

The RAG Service enables users to pose natural-language questions about any book in the library and receive accurate, page-cited answers. As illustrated in Figure 4.6, rather than relying on a language model's parametric knowledge, the service first retrieves the most relevant passages from the book and provides them as explicit context to the LLM, grounding its response in the actual text.

4.5.1 Architecture

The service is built on the **LangChain** framework with **Groq's Llama-3.3-70b-Versatile** as the generation model and Pinecone as the vector retrieval backend. The pipeline consists of three sequential components: a retriever, a chain, and a streaming layer.

Table 4.7. RAG Service Component Overview

Component	Technology	Responsibility
Retriever	PineconeBookRetriever (LangChain BaseRetriever)	Encodes the user query and fetches top-K passages from Pinecone by ISBN namespace
Chain	LangChain Expression Language (LCEL) retrieval pipeline + PromptTemplate	Formats retrieved passages as context and submits them with the question to the LLM
LLM	Llama-3.3-70b-Versatile (temperature = 0.1)	Generates a grounded answer citing specific page numbers from the provided context (accessed via Groq API)
Streaming	Server-Sent Events (SSE) + asyncio Queue	Delivers tokens to the client incrementally as the LLM produces them
Cache	LRU cache — 256 entries	Avoids rebuilding the LangChain chain object on every request per ISBN

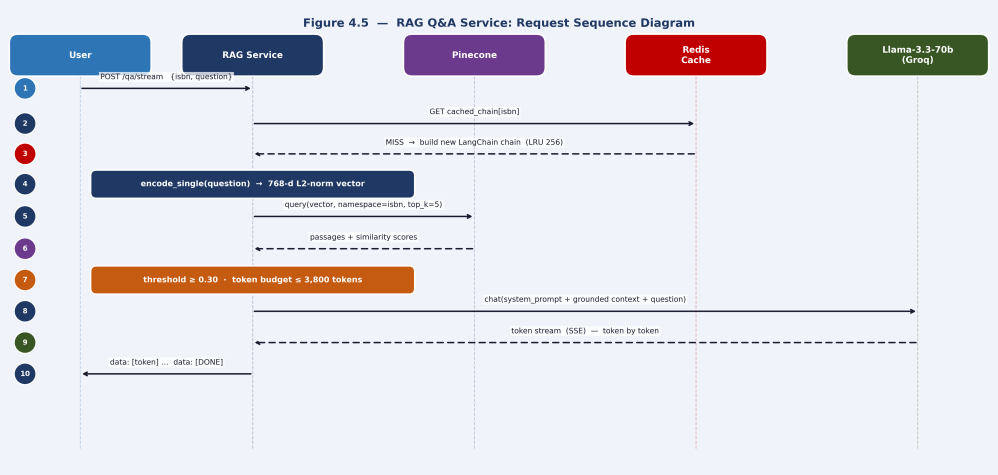


Figure 4.6. RAG Q&A Sequence Diagram — from User Request to Streamed Answer

4.5.2 Retriever — Similarity Threshold Filtering

The `PineconeBookRetriever` encodes the user's question using the same `all-mpnet-base-v2` model used during ingestion — query and document vectors must occupy the same semantic space for similarity search to be meaningful. The resulting 768-dimensional query vector is submitted to Pinecone for ANN search, retrieving the top 5 candidate passages (`RAG_TOP_K = 5`) within the namespace corresponding to the requested book's ISBN (steps 3–4 in Figure 4.6).

A similarity threshold of **0.30** is applied to the retrieved results: any passage scor-

ing below this threshold is discarded before context assembly (step 5 in Figure 4.6). This mechanism serves as a **hallucination guard** — without it, a model receiving weakly-related passages may generate plausible-sounding answers not supported by the book. If no passages exceed the threshold, the chain returns a standardised insufficient-information message.

4.5.3 Chain — Grounded Prompting & Token Budget

The RAG pipeline is implemented as a LangChain Expression Language (LCEL) chain composed of a retriever, prompt template, and LLM (the now-deprecated `RetrievalQA` wrapper was replaced with this composable equivalent). It follows the "stuff" strategy, concatenating all retrieved passages into a single context block submitted to Llama-3.3-70b-Versatile in one request. A custom system prompt enforces three strict behavioural constraints on the model:

- Answer using **only** the provided excerpts — no external or parametric knowledge is permitted.
- Cite the page number(s) from which each piece of information was drawn.
- Respond with a standardised message if the provided excerpts are insufficient to answer the question.

Listing 4.2. RAG System Prompt Definition

```
SYSTEM_PROMPT = (
  "You are an intelligent reading assistant for the Smart AI
     Library. "
  "Answer using ONLY the excerpts provided below. "
  "If the excerpts are insufficient, respond: "
  "'I don't have enough information from this book to answer
     that.' "
  "Always cite the page number(s) you drew information from."
)
```

A token budget guard (step 6 in Figure 4.6) truncates the assembled context to a maximum of 3,800 tokens using `tiktoken` (`cl100k_base` encoding), preventing context-overflow API failures. Chain objects are cached per ISBN via `functools.lru_cache` (capacity 256), avoiding rebuild overhead on repeated requests for the same book.

4.5.4 Streaming — Server-Sent Events (SSE)

To improve perceived responsiveness, the service delivers Llama-3.3-70b-Versatile's response token-by-token via Server-Sent Events (SSE), as shown in steps 8–9 of Figure 4.6. This allows users to begin reading the answer while it is still being generated — par-

ticularly valuable for detailed queries producing several paragraphs of output.

An `asyncio Queue` acts as an in-process message bus between the LangChain streaming callback and the SSE generator coroutine. As the model produces each token, the `QueueCallback` places it in the queue; the generator dequeues and yields formatted SSE frames to the HTTP response. A 15-second inter-token timeout terminates the stream if the model stalls, preventing client connections from hanging indefinitely.

Table 4.8. SSE Protocol Events in the RAG Streaming Endpoint

SSE Event	Meaning
`data: <token>`	A generated token — the client appends it to the displayed answer
`data: [DONE]`	Generation complete — the client should close the event stream
`data: [TIMEOUT]`	No token received within 15 seconds — stream terminated by server
`data: [ERROR]`	An exception occurred during chain execution — stream aborted

4.6 Recommendation Service — Hybrid NCF + CBF

The Recommendation Service delivers personalised book recommendations by combining two complementary AI techniques: Neural Collaborative Filtering (NCF), which learns from user behaviour, and Content-Based Filtering (CBF), which reasons over book content semantics. A hybrid fusion layer blends the two signals with weights that adapt dynamically based on how much interaction history the user has accumulated.

4.6.1 Neural Collaborative Filtering (NCF)

The NCF model is a PyTorch implementation of the Generalised Matrix Factorisation + MLP architecture (He et al., 2017). It maintains two separate embedding tables for users and items — one for the GMF branch and one for the MLP branch — allowing each pathway to learn complementary interaction patterns. The GMF branch computes an element-wise product of user and item embeddings, capturing linear co-occurrence signals. The MLP branch concatenates the embeddings and passes them through three fully-connected layers [128, 64, 32] with ReLU activations and 0.2 dropout, capturing non-linear interaction patterns. The outputs of both branches are concatenated and passed through a final sigmoid layer to produce a relevance score in $[0, 1]$.

Table 4.9. NCF Model Hyperparameters

Hyperparameter	**Value**
Embedding Dimension (emb_dim)	64
MLP Hidden Layers	[128, 64, 32]
Dropout Rate	0.2
Output Activation	Sigmoid $\rightarrow$ score in $[0, 1]$
Weight Initialisation	Normal (std=0.01) for embeddings; Xavier uniform for linear layers

The model is updated incrementally via an Online Updater that consumes user interaction events from the Kafka topic `user.events`. Events classified as positive signals (`read_start`, `bookmark`, `rating_4`, `rating_5`, `purchase`) trigger a single SGD step on the user embedding layers only (learning rate = 5×10^{-4}), keeping item embeddings stable while personalising user representations in near real-time.

4.6.2 Content-Based Filtering (CBF)

The CBF layer uses Pinecone's `synopses` namespace, which stores one synopsis vector per book. Each synopsis vector is the L2-normalised average of all chunk vectors for that book, computed and upserted by the Embedding Service after ingestion. To find books similar to a seed title, the CBF layer fetches the seed's synopsis vector and queries Pinecone for the closest neighbours by cosine similarity. If a synopsis vector is not yet available, the layer falls back to averaging the seed book's chunk vectors from its per-ISBN namespace.

4.6.3 Hybrid Fusion and User Tiers

The final recommendation score is a weighted linear blend of the NCF and CBF scores:

$$\text{score} = \alpha \times \text{CF} + (1 - \alpha) \times \text{CBF} \tag{4.1}$$

The weight α is determined by the user's interaction tier, which is computed from the number of reading events recorded in the database:

Table 4.10. Hybrid Fusion Alpha Values by User Tier

Tier	**Condition**	**Alpha (α)**
New	Fewer than 5 interactions	0.10 — mostly content-based
Returning	5 to 49 interactions	0.55 — balanced blend
Power	50 or more interactions	0.80 — mostly collaborative

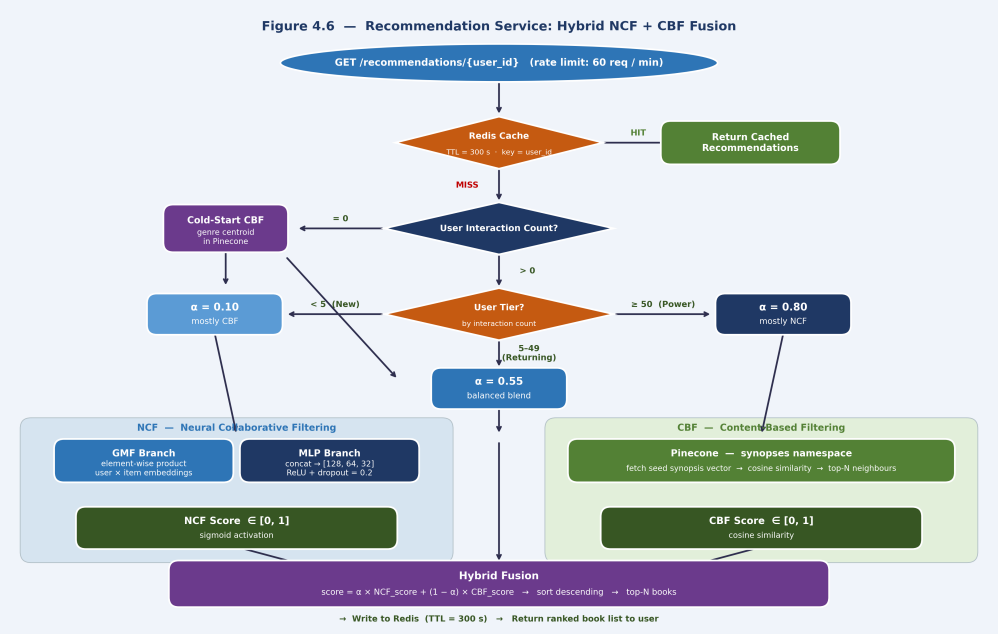


Figure 4.6 — Recommendation Service: Hybrid NCF + CBF Fusion
GET /recommendations/{user_id} (rate limit: 60 req / min)
Redis Cache
TTL = 300 s · key = user_id
HIT
Return Cached Recommendations
MISS
User Interaction Count?
= 0
Cold-Start CBF
genre centroid in Pinecone
> 0
User Tier?
by interaction count
< 5 (New)
α = 0.10
mostly CBF
≥ 50 (Power)
α = 0.80
mostly NCF
5–49 (Returning)
α = 0.55
balanced blend
NCF — Neural Collaborative Filtering
GMF Branch
element-wise product
user × item embeddings
MLP Branch
concat → [128, 64, 32]
ReLU + dropout = 0.2
NCF Score ∈ [0, 1]
sigmoid activation
CBF — Content-Based Filtering
Pinecone — synopses namespace
fetch seed synopsis vector → cosine similarity → top-N neighbours
CBF Score ∈ [0, 1]
cosine similarity
Hybrid Fusion
score = α × NCF_score + (1 − α) × CBF_score → sort descending → top-N books
→ Write to Redis (TTL = 300 s) → Return ranked book list to user

Figure 4.7. Recommendation Service — Hybrid NCF + CBF Fusion

This tiered approach solves the cold-start problem: new users with little interaction history receive content-driven recommendations based on book semantics, while power users benefit primarily from collaborative signals learned from their accumulated behaviour. For users with no interaction history at all, a separate `/recommendations/cold-start` endpoint uses CBF only, optionally filtering by genre, to surface popular or genre-representative titles.

Recommendations are cached in Redis for 5 minutes (`REDIS_TTL_REC = 300` seconds) per user and are subject to a rate limit of 60 requests per minute per user, enforced via a Redis sliding-window counter.

4.7 Mobile Client Integration — Lesson Assistant and Lightweight Fallback Pathway

Alongside the Pinecone- and Kafka-based pipeline described in Sections 4.1 to 4.6, the Learnova mobile client (built with Flutter) integrates a second, lightweight AI pathway implemented directly at the Laravel API layer. This pathway serves two purposes that fall outside the scope of the book-centric vector pipeline: first, it powers an AI assistant for course lessons, a video-based content type that is never ingested into the Pinecone index; second, it acts as a fallback for book chat and summarisation when a book cannot be resolved against the ISBN-indexed vector store. Where the primary pipeline grounds its answers in retrieved, embedding-ranked passages, this pathway grounds its answers by passing client-supplied text — a video transcript or client-extracted PDF text — directly into the prompt sent to a lightweight hosted model, trading persistent

semantic indexing for simplicity and immediate availability.

4.7.1 Course Lesson Assistant — Whisper Transcription and Contextual Chat

Each course lesson screen exposes a pulsing, floating AI button (`CourseAIFloatingActionButton`). On tap, the client first issues a POST request to `/lessons/{id}/transcribe` before opening the chat sheet. The route checks whether the lesson already has a cached transcript; if so, it is returned immediately, avoiding repeated transcription of the same video. Otherwise, the server locates the lesson's video file on disk and invokes OpenAI's Whisper speech-to-text model (the "small" checkpoint) as a local Python subprocess, writing the resulting text file to a transcripts directory and persisting its contents to the lesson record. Transcription is treated as non-blocking on the client: if the request fails or times out, the chat sheet still opens, simply without lesson-transcript context.

The chat sheet itself (`InlineChatSheet`) follows the same Learnova AI visual identity used across the platform: a draggable bottom sheet with snap points at 40%, 72%, and 93% of the screen height, a header identifying the assistant, and a scrollable message list rendered through shared `ChatBubble` and `TypingIndicator` widgets. To reduce friction for common study tasks, four quick-reply chips — Summarize, Make Exam, Explain, and Key Terms — send pre-written instructional prompts without requiring the learner to type them out. A history panel, backed by GET and DELETE calls to `/chat/conversations`, lets users revisit or remove earlier conversations.

This implementation currently has two portability limitations worth noting for future iterations: the path to the Python interpreter used to invoke Whisper is hardcoded to a local development machine rather than read from environment configuration, and transcription runs synchronously within the HTTP request lifecycle (guarded by a five-minute PHP time limit) rather than as a queued background job. Migrating transcription to an asynchronous queue, consistent with the background-job pattern already used by the Ingestion Service in Section 4.1.3, is identified as near-term future work.

4.7.2 Book Chat and Summarisation — Fallback Pathway

The book-level counterparts, `BookInlineChatSheet` and `BookSummarizeSheet`, reuse the same chat UI and branding but talk to two dedicated Laravel routes, `/books/{id}/chat` and `/books/{id}/summarise`, that bypass the Pinecone-based Search and RAG services entirely. Rather than retrieving passages from a vector index, the client extracts the book's text locally and sends it as a `pdf_text` field in the request body; the server interpolates this text directly into the system prompt, effectively performing retrieval and grounding at the application layer instead of through the Embedding and Search Services described in Sections 4.2 and 4.3. Conversations created through this route

are persisted in the same relational `ChatConversation` and `ChatMessage` tables used by the lesson assistant, independent of the ISBN-keyed vector namespace used by the RAG service.

Summarisation explicitly branches on data availability at the client. When a book record has a populated ISBN, the summarise sheet calls `/ai/summarise`, routing the request into the FLAN-T5-large map-reduce pipeline described in Section 4.4. When no ISBN is available — for instance, a book uploaded without matching bibliographic metadata — the client instead calls `/books/{id}/summarise`, which asks the fallback model directly for a short (5–7 bullet point) or detailed summary, using any client-supplied PDF text as additional context when present. Both fallback routes call the OpenRouter API using the `openai/gpt-oss-20b:free` model, in contrast to the Groq-hosted Llama-3.3-70B-Versatile model used by the primary RAG service in Section 4.5.

Table 4.11. Lightweight Fallback Pathway — Endpoints and Backing Models

Endpoint	**Backing Model**	**Purpose**
POST /lessons/{id} /transcribe	Whisper (small, local subprocess)	Transcribes a lesson video to text; result is cached on the lesson record
POST /books/{id}/chat	openai/gpt-oss-20b (OpenRouter)	Conversational Q&A over a book, grounded by client-supplied PDF text
POST /books/{id} /summarise	openai/gpt-oss-20b (OpenRouter)	Short or detailed summary generated directly from client-supplied PDF text
POST /ai/summarise	FLAN-T5-large (Section 4.4)	Primary summarisation route, used when the book has an indexed ISBN

4.7.3 Shared Mobile Client Architecture

Both pathways are surfaced through a common `ChatCubit`, following the BLoC state-management pattern used throughout the Flutter application. The cubit exposes `startChatWithLesson`, `startChatWithBook`, `sendMessage`, and `sendMessageForBook` methods, so that the presentation layer — the chat sheets and quick-reply chips — remains agnostic to which backend route ultimately handles a given request. All AI entry points across the app, whether triggered from a lesson's floating button or a book's chat and summarise sheets, share the same visual language: the `AppColors.myGradient` identity, the "Learnova AI" header, and the same draggable-sheet interaction pattern.

This shared interface is a deliberate design choice: from the learner's point of view, asking a question about a lesson and asking a question about a book feel identical,

even though one answer may be retrieval-grounded and page-cited (Section 4.5) while the other is generated directly from supplied context with no citation mechanism. This trade-off is discussed further in Section 4.7.4.

4.7.4 Design Trade-offs

Routing lesson assistance and ISBN-less book interactions through a lightweight, direct-prompting pathway rather than extending the full RAG pipeline to cover them involves a deliberate trade-off:

- **Advantages:** no dependency on Pinecone or Kafka availability, lower operational complexity for content that is not part of the core book catalogue, and immediate support for a content type — lesson video — that the ingestion pipeline was not designed to index.
- **Disadvantages:** answers are not anchored to a persistent semantic index, so this content cannot participate in cross-book search (Section 4.3) or influence the Recommendation Service (Section 4.6); there is no equivalent of the RAG service's similarity-threshold filtering or page-level citation mechanism (Sections 4.5.2 and 4.8.2); and reliance on a smaller, free-tier hosted model may yield lower answer quality than the 70-billion-parameter model used by the primary pipeline.

The fallback pathway is therefore best understood as a pragmatic extension of AI coverage to content and edge cases outside the primary pipeline's scope, rather than a replacement for it — a distinction made explicit to users only implicitly, through the absence of citations in fallback-generated answers.

4.8 Ethical Considerations

The deployment of AI-powered features in a library context carries responsibilities beyond technical correctness. The following ethical principles were explicitly considered during the design and implementation of the platform's AI subsystems.

4.8.1 Hallucination Mitigation

Large language models can generate factually incorrect information with high confidence — a phenomenon termed *hallucination*. In a library context, a hallucinated answer could seriously mislead researchers or students. The RAG service addresses this through two complementary mechanisms: the similarity threshold filter (Section 4.5.2) rejects weakly-related passages before they reach the LLM, and the system prompt (Section 4.5.3) explicitly forbids the model from drawing on knowledge outside the provided excerpts. When no passages exceed the threshold, the system returns a standardised

message rather than a speculative answer.

4.8.2 Transparency and Source Attribution

All RAG Q&A answers include explicit page-number citations from the source documents returned by the retriever. Users can verify any claim by consulting the original book at the cited page. This design treats AI-generated answers as a navigational aid rather than an authoritative endpoint, preserving the user's ability to critically evaluate information and maintaining trust in the library system.

4.8.3 Content Scope and Copyright

The platform processes only book files uploaded by authenticated library administrators, who are assumed to have verified copyright compliance before upload. The system does not crawl external sources, generate original creative content, or reproduce large verbatim passages. Generated summaries are abstractive paraphrases intended to assist comprehension, not to substitute for reading the original work or to circumvent copyright restrictions.

4.8.4 Model Bias and Limitations

The pre-trained models used in this platform (`all-mpnet-base-v2`, `FLAN-T5-large`, `Llama-3.3-70b-Versatile`, and `cross-encoder/ms-marco-MiniLM-L-6-v2`) were not fine-tuned on library-domain data. As a result, they may perform less accurately on highly specialised technical content or non-English texts. The platform currently supports English-language content; extension to Arabic and other languages is identified as a planned future enhancement. Users are advised to treat AI-generated outputs as supplementary tools rather than definitive references. The lightweight fallback pathway described in Section 4.7 carries an additional caveat: it relies on a smaller, free-tier hosted model (`openai/gpt-oss-20b`) rather than the 70-billion-parameter model used by the primary RAG service, and it does not apply the same similarity-threshold or citation safeguards described in Section 4.8.1, so its outputs should be treated with correspondingly more caution.

Chapter Summary

This chapter documented the AI subsystems of the AIGP Smart Library platform. The primary pipeline consists of six services built around a persistent Pinecone vector index. The Ingestion Service (Section 4.1) accepts PDF and ePub files, extracts and cleans text, and publishes 512-token sliding-window chunks with 50-token overlap to Kafka for downstream processing. The Embedding Service (Section 4.2) uses

`all-mpnet-base-v2` to produce L2-normalised 768-dimensional vectors, stored in Pinecone via a Kafka-driven batch pipeline with exponential back-off retries and dead-letter queue fault tolerance. The Search Service (Section 4.3) implements a two-stage retrieval pipeline — bi-encoder ANN retrieval over 50 candidates followed by cross-encoder/ms-marco-MiniLM-L-6-v2 re-ranking — returning the top 10 results with Redis caching. The Summarisation Service (Section 4.4) applies a FLAN-T5-large map-reduce pipeline to handle full-book content, producing both textual summaries and JSON mind maps with a rule-based fallback. The RAG Q&A Service (Section 4.5) combines Pinecone retrieval with Llama-3.3-70b-Versatile (Groq) generation, enforcing grounded output through a similarity threshold of 0.30, a 3,800-token budget guard, and a strict system prompt, while delivering responses incrementally via SSE streaming. The Recommendation Service (Section 4.6) fuses NCF collaborative filtering with CBF semantic similarity using a tiered alpha weighting (0.10 / 0.55 / 0.80) that adapts to each user's interaction history, with online model updates driven by Kafka user events. Section 4.7 complemented this pipeline with a lightweight, directly-prompted AI pathway at the mobile client's Laravel layer, extending assistance to course-lesson video content via Whisper transcription and providing a fallback chat and summarisation route for books outside the vector-indexed catalogue.

Table 4.12. AI Services Comparison

Service	**AI Model(s)**	**Input**	**Output**	**Latency**
Ingestion	(none — rule-based)	PDF / ePub file	Chunks on Kafka book.chunks	Async / background
Embedding	all-mpnet-base-v2	Text chunks (Kafka)	768-d vectors in Pinecone	Async / background
Search	all-mpnet-base-v2 + cross-encoder/ms-marco-MiniLM-L-6-v2	Natural-language query	Top 10 ranked passages	Low latency (<200 ms typical)
Summarisation	FLAN-T5-large	Book ISBN	Summary text + Mind Map JSON	Seconds to minutes
RAG Q&A	all-mpnet-base-v2 + Llama-3.3-70b-Versatile (Groq)	Question + ISBN	Streamed answer + citations	Near real-time first token (SSE)
Recommendation	NCF (PyTorch) + CBF (Pinecone)	User ID	Ranked book list	Low latency (Redis cached)
Lesson Assistant / Fallback	Whisper (small) + openai/gpt-oss-20b (OpenRouter)	Video transcript or PDF text + question	Contextual answer (no citations)	Near real-time (non-streaming)

Together, these services realise the platform's core value proposition: transforming a static digital book collection into a dynamic, queryable knowledge base where answers are grounded in the source text, attributed to specific pages, and always verifiable by the user.

Chapter 5

Flutter Mobile Implementation

Contents

3.1 Overview 70
3.2 Home, Search & Profile Screens 71
3.3 Authentication & Onboarding Screens 72
3.4 Book Detail Screen 74
3.5 Author Profile Screen 78
3.6 Instructor Profile Screen 81
3.7 Course Detail Screen 83
3.8 Media Player — Lesson & Book Audio Player 85
3.9 Favourites Management System 88
3.10 AI Chatbot — General & Contextual 90
3.11 BLoC / Cubit State Machine 93
3.12 REST API Endpoint Map 94
3.13 Navigation and Routing (go_router) 95
3.14 Error Handling and UX Patterns 96
3.15 Overall Challenges and Solutions 98

Flutter Dart BLoC Equatable Easy Localization Flutter TTS

Figure 5.1. Technology Stack Used in the Learnova Flutter Mobile App

5.1 Overview

The Learnova Flutter mobile application delivers a comprehensive, responsive, and interactive learning experience across two primary domains: Books (Audiobooks & E-books) and Courses. The mobile client communicates exclusively with the Learnova Laravel REST API, authenticated via Laravel Sanctum tokens persisted in `SharedPreferences`.

The application follows a clean-architecture pattern divided into four clearly separated layers, each with a single responsibility, as illustrated in Figure 5.2.

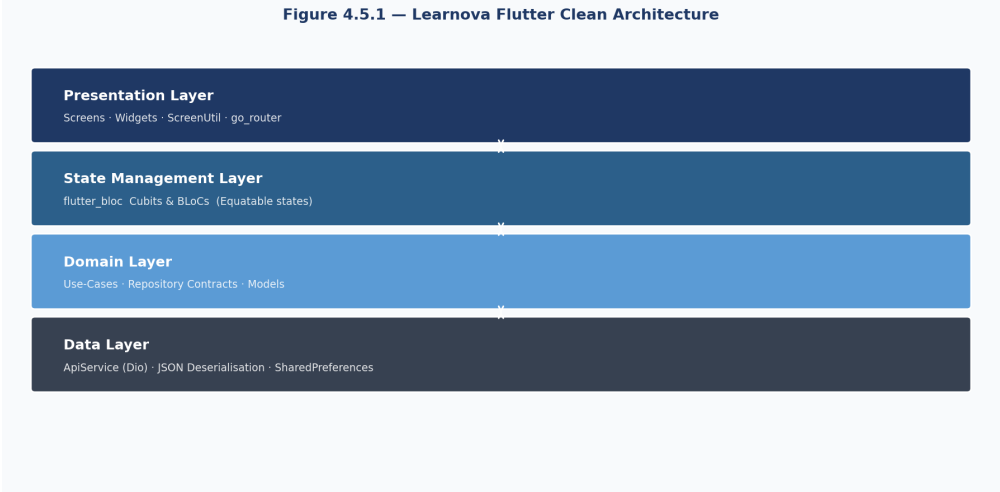


Figure 4.5.1 — Learnova Flutter Clean Architecture
Presentation Layer
Screens · Widgets · ScreenUtil · go_router
State Management Layer
flutter_bloc Cubits & BLoCs (Equatable states)
Domain Layer
Use-Cases · Repository Contracts · Models
Data Layer
ApiService (Dio) · JSON Deserialisation · SharedPreferences

Figure 5.2. Learnova Flutter Clean Architecture Layers

- **Data Layer** — `ApiService` classes built on Dio, JSON deserialisation into Model classes, and `SharedPreferences` persistence.
- **Domain Layer** — Use-case contracts and repository interfaces decoupled from the framework.
- **State Layer** — `flutter_bloc` Cubits and BLoCs managing reactive state with `Equatable` states.
- **Presentation Layer** — Widgets and Screens built with `flutter_screenutil` for device-responsive sizing.

All authenticated endpoints share the base URL `http://<server>/api` and require an `Authorization: Bearer <sanctum_token>` header.

Table 5.1. Common Packages Used Across All Features

Package	Purpose
flutter_bloc ^9.1.1	State management (Cubit & BLoC)
equatable ^2.0.8	Value equality for state classes
dio ^5.9.1	HTTP client, interceptors, error handling
shared_preferences ^2.5.3	Persist auth token & user preferences
flutter_screenutil ^5.9.3	Responsive sizing (sp / w / h)
cached_network_image ^3.4.1	Cached remote images with fade-in
go_router ^17.0.0	Declarative navigation & deep linking
syncfusion_flutter_pdfviewer ^31.1.19	Inline PDF rendering
video_player ^2.10.1	Video / audio playback
flutter_tts ^4.2.5	Text-to-speech for audiobooks
flutter_markdown ^0.7.7	Render AI markdown responses
easy_localization ^3.0.8	Arabic / English localisation
connectivity_plus ^6.1.5	Network connectivity monitoring for offline detection
shimmer ^3.0.0	Skeleton loading placeholders

5.2 Home, Search & Profile Screens

Before detailing each feature screen, this section presents the three screens a user encounters most frequently: the Home feed, global Search, and the user's own Profile.

5.2.1 Home Screen

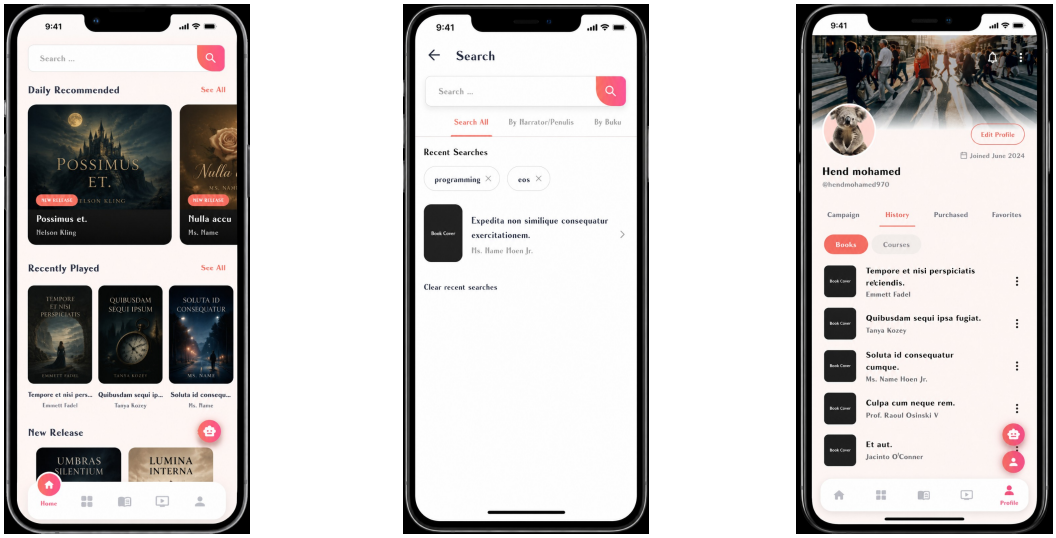
The `HomeScreen` is the application's landing screen after login, presenting a personalised, AI-driven feed organised into horizontally scrolling sections — Daily Recommended, Recently Played, and New Release — each populated from the Recommendation Service described in the previous chapter. A persistent bottom navigation bar (Home, Discover, Books, Courses, Profile) and a floating AI assistant button remain accessible throughout.

5.2.2 Search Screen

The `SearchScreen` provides tabbed search across the entire catalogue (Search All, By Narrator/Author, By Book), backed by the Search Service's semantic ranking. Recent searches are persisted locally and displayed as removable chips above the results list.

5.2.3 Profile Screen

The `ProfileScreen` displays the authenticated user's avatar, display name, and join date, alongside tabs for Campaign, History, Purchased, and Favourites content, filterable between Books and Courses.



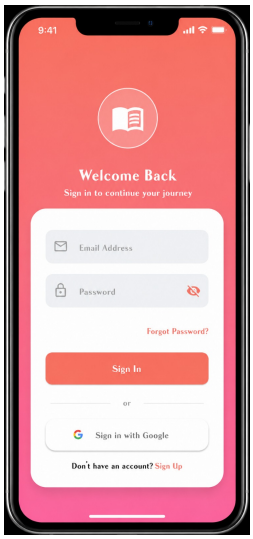
(a) Home **(b)** Search **(c)** Profile

Figure 5.3. Home, Search, and Profile Screens

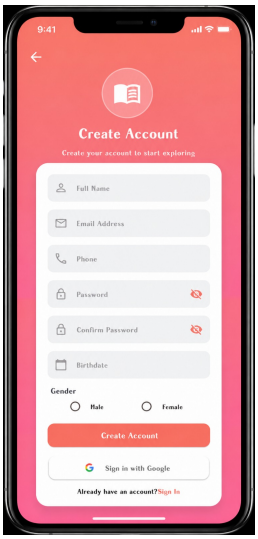
5.3 Authentication & Onboarding Screens

Authentication follows a standard email/password flow via Laravel Sanctum, complemented by Google social login and an OTP-based password recovery flow. All authentication screens share the same gradient visual identity — a warm coral-to-pink gradient with a centred logo mark — establishing the application's brand from the very first screen a user sees.

- **LoginScreen** — email/password fields, "Forgot Password?" link, and a "Sign in with Google" button.
- **RegisterScreen** — full name, email, phone, password, confirm password, birthdate, and gender selection, with the same Google sign-in option.
- **ForgotPasswordScreen** — collects the user's email and triggers `/api/forgot-password/send`
- **VerifyCodeScreen** — a 4-digit OTP input, validated against `/api/forgot-password/verify`
- **ResetPasswordScreen** — new password and confirmation fields, submitted to `/api/forgot-password/reset`.

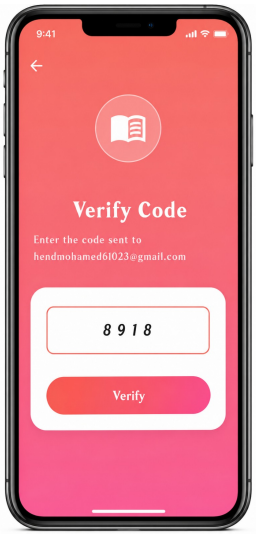


(a) Login



(b) Create Account

(c) Forgot Password



(d) Verify OTP Code

(e) Reset Password

Figure 5.4. Authentication Screens: Login, Registration, and OTP-Based Password Recovery

5.3.1 Onboarding — Topic Preferences

Immediately after a user registers (or logs in for the first time), the `OnboardingScreen` presents a searchable grid of topic categories — Novel, Science & Tech, History, Biographies, Self Help, Programming, Inspirational, Comedy, Horror, Kids, and others — sourced from `GET /api/topics`. The user selects one or more topics of interest and taps "Continue" to submit the selection via `POST /api/topics/save`, personalising the Daily Recommended and New Release sections of the Home screen shown in Figure 5.3 from the very first session. As discussed in Section 6.5 of the Testing chapter, this endpoint originally expected numeric category IDs under the key `category_ids`; the client-side payload was corrected accordingly after this mismatch surfaced during

manual API testing.

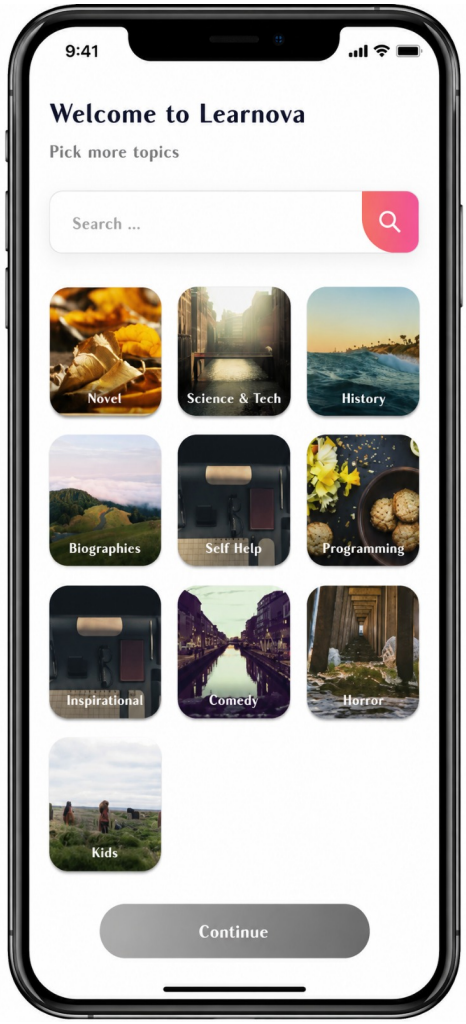


Figure 5.5. OnboardingScreen — Topic Preference Selection Grid

5.4 Book Detail Screen

5.4.1 Objective

The `BookDetailScreen` provides a rich reading and listening experience for a single book, combining PDF viewing, text-to-speech audio playback, AI-powered summarisation, and an interactive contextual chatbot — all within one cohesive, tabbed interface.

5.4.2 UI/UX Implementation

- Gradient hero header with book cover image, title, author name, category badge, and star rating.
- Tabbed layout: Overview, Read (PDF), Listen (TTS), AI Tools.
- **Overview tab** — synopsis text, metadata row (pages, language, year), similar

books horizontal scroll.

- **Read tab** — `SfPdfViewer` with page counter, zoom pinch gesture, and reading progress persistence.
- **Listen tab** — custom audio player bar: play/pause, seek slider, 0.5×–2× speed, chapter list drawer.
- **AI Tools tab** — two cards: "Summarise Book" (short / detailed) and "Chat with Book".
- Floating action row: favourite heart (`FavoritesBloc`), share button, remove-from-history.

Figure 5.6 illustrates the complete feature flow of `BookDetailScreen` from user entry to the various sub-features, and Figure 5.7 shows the resulting screen alongside its audio playback counterpart.

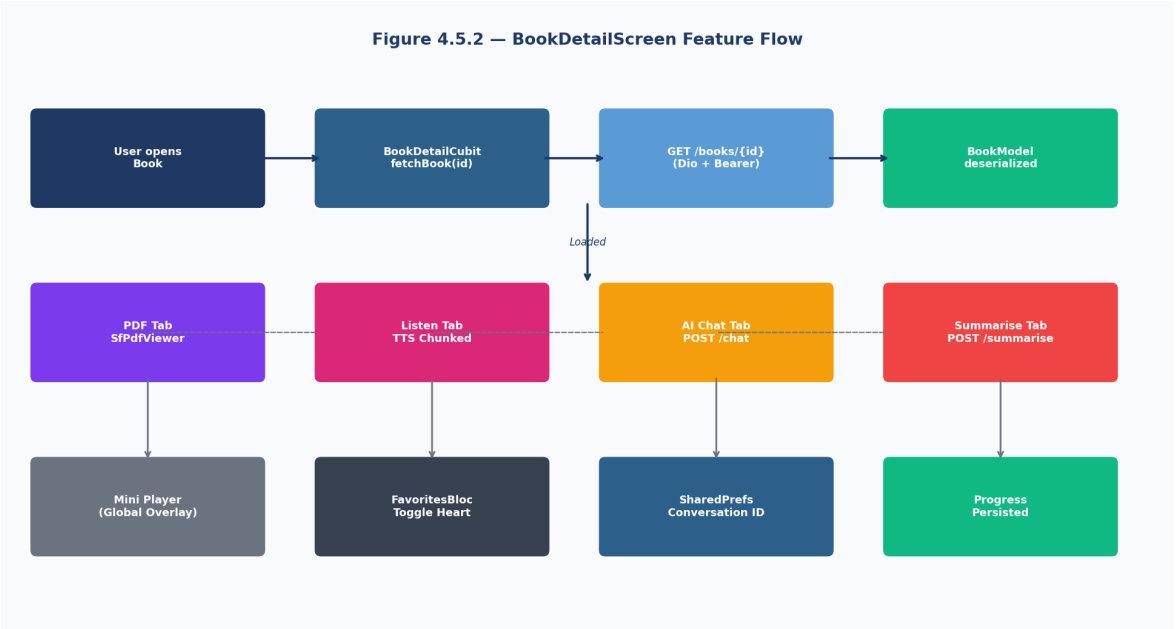
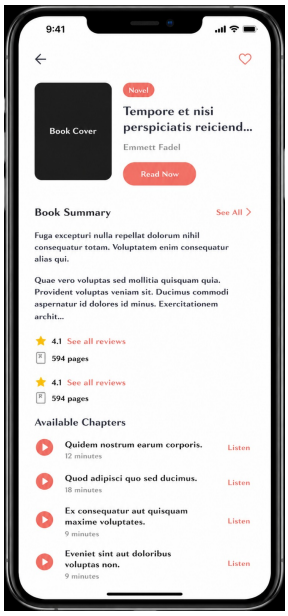
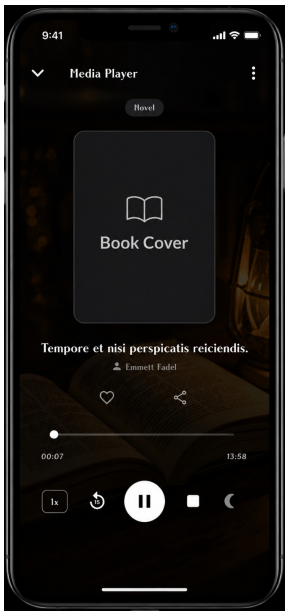


Figure 5.6. BookDetailScreen Feature Flow Diagram



(a) Book Detail — Overview tab



(b) Book Audio Player — Listen tab

Figure 5.7. BookDetailScreen and its Audio Playback Interface

5.4.3 API Endpoints

Table 5.2. BookDetailScreen API Endpoints

Method + URL	Description
GET /books/{id}	Fetch book metadata, PDF URL, audio URL, cover image
POST /books/{id}/favorite	Toggle favourite status — returns updated `is_favorite`
POST /books/{id}/listen	Log a listen event for analytics / history
GET /books/{id}/share	Returns a shareable deep-link URL
POST /books/{id}/chat	Send user message → receive AI reply
POST /books/{id}/summarise	Request short or detailed AI summary
DELETE /books/{id}/history	Remove book from user reading history
POST /books/{id}/reviews	Submit a star rating + text review
GET /books/{id}/reviews	Paginated list of reviews for the book

5.4.4 Request and Response Samples

Listing 5.1. GET /books/id — Sample Response

```
{ "id": 12, "title": "Clean Code",
  "author": { "id": 3, "name": "Robert C. Martin" },
  "category": { "id": 2, "name": "Technology" },
  "pdf_url": "https://cdn.example.com/books/clean_code.pdf",
```

```
  "cover_image": "https://cdn.example.com/covers/clean_code.jpg",
  "description": "A handbook of agile software craftsmanship...",
  "rating": 4.7, "is_favorite": false, "pages": 431 }
```

Listing 5.2. GET /books/id/share — Sample Response

```
{ "share_url": "https://learnova.app/books/12" }
```

Listing 5.3. POST /books/id/chat — Request and Response

```
// Request
{ "message": "What is the main theme of chapter 3?",
  "history": [ {"role":"user","content":"..."},
     {"role":"assistant","content":"..."} ],
  "conversation_id": 5, "pdf_text": "..." }

// Response 200 OK
{ "id": "abc123", "message": "Chapter 3 focuses on functions...",
  "role": "assistant", "conversation_id": 5 }
```

5.4.5 Models

Table 5.3. BookDetailScreen Model Classes

Model Class	Key Fields
BookModel	id, title, description, pdfUrl, coverImage, rating, isFavorite, pages, author, category
AuthorModel	id, name, bio, imageUrl, monthlyListeners
ReviewModel	id, userId, userName, rating, comment, createdAt
ChatMessageModel	id, role (user/assistant), content, conversationId

5.4.6 State Management

- `BookDetailCubit` — fetches book data, toggles favourite, logs listen event. States: `BookDetailInitial` | `BookDetailLoading` | `BookDetailLoaded` | `BookDetailError`.
- `BookChatCubit` — manages message list, sends messages, appends AI responses optimistically.
- `BookSummaryCubit` — requests summary; emits `SummaryLoading` | `SummaryLoaded` | `SummaryError`.
- `FavoritesBloc` (global root) — updated on toggle; keeps heart icon in sync across all screens.

5.4.7 Integration Steps

1. `initState()` calls `BookDetailCubit.fetchBook(id)` → emits Loading → shimmer skeleton shown.
2. On `BookDetailLoaded` state, build tabs and pass `pdfUrl` to `SfPdfViewer`.
3. PDF text extracted client-side via `syncfusion_flutter_pdf` in a `compute()` Isolate, cached in state.
4. Extracted `pdf_text` ($\leq$ 8,000 chars) sent in `POST /chat` and `POST /summarise` for RAG grounding.
5. TTS: text split into paragraphs $\leq$ 4,000 chars; `flutter_tts.onComplete` callback chains the next chunk.
6. Favourite toggle dispatches to `FavoritesBloc` → optimistic update → API call → revert on error.

5.4.8 Challenges and Solutions

Table 5.4. BookDetailScreen Challenges and Solutions

Challenge	Solution Applied
flutter_tts timeout on books > 50 pages	Text split into $\leq$ 4,000-char paragraph chunks; `onComplete` callback auto-advances to next chunk
PDF parsing blocking the UI thread	`syncfusion_flutter_pdf` parsing moved to an Isolate via `compute()`; result cached in Cubit state
Chat history lost on hot-restart	`conversation_id` stored in `SharedPreferences`; reloaded via `GET /chat/conversations/{id}` on screen open
PDF text causing 413 Payload Too Large	Client truncates `pdf_text` to the first 8,000 characters before sending in the request body

5.5 Author Profile Screen

5.5.1 Feature Name

`AuthorInfoScreen`

5.5.2 Objective

`AuthorInfoScreen` displays a detailed author biography, popularity metrics (monthly listeners), popular and latest book releases, and similar authors — enabling users to discover and follow content creators.

5.5.3 UI/UX Implementation

- Gradient hero header: author photo, name, monthly listeners counter, follow/unfollow button.
- "Popular Releases" horizontal scroll — `BookCard` widgets with favourite heart icon overlay.
- "Latest Releases" horizontal scroll — reusable `HorizontalBookList` widget.
- "Similar Authors" section — `ArtistCard` chips with inline follow toggle.
- Optimistic follow count: UI increments/decrements immediately; API confirms asynchronously.

Figure 5.8. AuthorInfoScreen — Biography, Popular Releases, and Follow Button

5.5.4 API Endpoints

Table 5.5. AuthorInfoScreen API Endpoints

Method + URL	Description
GET /authors/{id}/profile	Full author profile including releases & similar authors
POST /favorites/author/{id}	Toggle author favourite (follow/unfollow)
GET /authors/{id}/similar	Paginated list of similar authors

5.5.5 Sample Response — GET /authors/{id}/profile (200 OK)

Listing 5.4. GET /authors/id/profile — Sample Response

```
{ "id": 3, "name": "Robert C. Martin", "bio": "...",
  "image_url": "https://cdn.example.com/authors/uncle_bob.jpg",
  "monthly_listeners": 124500, "is_following": false,
  "popular_releases": [ { "id":12, "title":"Clean Code",
     "cover_image":"...", "rating":4.7 } ],
  "latest_releases":  [ { ... } ],
  "similar_artists":  [ { "id":5, "name":"Martin Fowler",
     "image_url":"..." } ] }
```

5.5.6 Models

`AuthorProfileModel` (id, name, bio, imageUrl, monthlyListeners, isFollowing, popularReleases, latestReleases, similarArtists), `AuthorReleaseItem` (id, title, coverImage, rating, isFavorite), and `SimilarArtistModel` (id, name, imageUrl, isFollowing).

5.5.7 State Management

Local `setState` for the follow toggle and loading overlay on the button. The global `FavoritesBloc` is reused for book favourite heart icons displayed within this screen.

5.5.8 Key Difference from Instructor Profile

`AuthorInfoScreen` uses book-specific models (`AuthorReleaseItem`) and the `HorizontalBookList` widget rather than `HorizontalCourseList`. Navigation targets `BookDetailScreen` instead of `CourseInfoScreen`.

5.6 Instructor Profile Screen

5.6.1 Feature Name

`InstructorProfileScreen`

5.6.2 Objective

`InstructorProfileScreen` provides students with detailed information about course instructors, including biography, teaching statistics, popular and latest courses, and similar instructors — building trust and facilitating instructor discovery.

5.6.3 UI/UX Implementation

- Gradient hero header: instructor photo, name, designation, real-time follower count.

- Follow button with optimistic update — follower count changes instantly while the API call is in-flight.
- "Popular Courses" horizontal scroll — `CourseCard` with favourite heart overlay.
- "Latest Courses" horizontal scroll — `HorizontalCourseList` reusable widget.
- "Similar Instructors" section — tappable chips navigating to another `InstructorProfileScree`

`InstructorProfileScreen` reuses the same `GradientProfileHeader` widget and layout shown for `AuthorInfoScreen` in Figure 5.8, with courses substituted for books; no separate screenshot is included here to avoid duplication.

5.6.4 API Endpoints

Table 5.6. InstructorProfileScreen API Endpoints

Method + URL	Description
GET /instructors/{id}/profile	Full instructor profile with courses & similar instructors
POST /favorites/instructor/{id}	Toggle follow / unfollow
GET /instructors/{id}/similar	List of similar instructors

5.6.5 Models

`InstructorProfileModel` (id, name, designation, bio, imageUrl, followerCount, isFollowing, popularCourses, latestCourses, similarInstructors), `InstructorReleaseItem` (converted to `CourseModel` inside the Cubit for reuse in shared widgets), and `SimilarInstructorIte` (id, name, imageUrl, isFollowing).

5.6.6 State Management

`InstructorProfileCubit` manages the profile fetch and follow/unfollow action, with states `InstructorProfileLoading` | `InstructorProfileLoaded` | `InstructorProfileError`. The global `FavoritesBloc` is shared for course favourite icons within this screen.

5.6.7 Integration Steps

1. The screen receives `instructorId` as a `String`; all entity IDs are normalised to `String` at the repository layer (see Section 5.13) and converted to `int` only where the API requires it.
2. `initState()` calls `InstructorProfileCubit.loadProfile(instructorId)`.
3. Release items are converted: `InstructorReleaseItem` → `CourseModel` before passing to `HorizontalCourseList`.

4. Tapping a similar instructor pushes another `InstructorProfileScreen(instructorId: item.id.toString())`.

5.6.8 Challenges and Solutions

Table 5.7. InstructorProfileScreen Challenges and Solutions

Challenge	Solution Applied
ID type mismatch (int vs String) across API responses	All IDs normalised to String at the repository layer using `.toString()`
Infinite navigation stack (instructor → similar → similar)	`go_router pushReplacement` prevents stack overflow on cyclic instructor navigation

5.7 Course Detail Screen

5.7.1 Feature Name

`CourseInfoScreen`

5.7.2 Objective

`CourseInfoScreen` provides comprehensive course information including description, curriculum, ratings, reviews, similar courses, and quick access to the instructor profile page.

5.7.3 UI/UX Implementation

- Course banner image with play-preview overlay button.
- Summary card: title, instructor name, price badge (Free / Paid), total duration, rating, enrolment count.
- Expandable description section with "Show More / Show Less" toggle.
- Curriculum accordion — lessons grouped by section, each row showing duration and lock icon.
- "Reviews" section: star distribution bar chart + paginated review cards.
- "Similar Courses" horizontal scroll using the shared `HorizontalCourseList` widget.
- "Instructor" card that navigates to `InstructorProfileScreen` on tap.

9:41
Business
Nesciunt eum et.
Anthony Connelly
Course Cover
Lesson: Quia repellat ad tempora necessit...
Course Summary
Reprehenderit asperiores qui possimus hic ut dicta nihil quisquam. Quidem in quia eos corrupti. Voluptate qui temporibus eum. Fugiat cum ea reprehenderit et odio veritatis quia illo.
Nulla omnis impedit accusamus doloribus mollitia. Voluptatum moles...
4.8 125 reviews
1202 students
1hr 30min
Available Lessons
Lesson: Quia repellat ad tempora necessitatibus.
13 minutes
Watch
Lesson: Non dolores itaque id aliquid mollitia.
2 minutes
Watch

Figure 5.9. CourseInfoScreen — Preview, Summary, and Available Lessons

5.7.4 API Endpoints

Table 5.8. CourseInfoScreen API Endpoints

Method + URL	Description
GET /courses/{id}	Full course detail with lessons & instructor info
GET /courses/home	Featured courses — used to populate similar courses widget
GET /courses/discover	Discovery data for recommended / related courses
POST /courses/{id}/progress	Update lesson watch progress percentage
POST /favorites/course/{id}	Toggle course favourite

5.7.5 Models

`CourseModel` (id, title, description, thumbnailUrl, instructorId, instructorName, price, isFree, rating, enrollmentCount, sections), `CourseSectionModel` (id, title, lessons: List⟨LessonModel⟩), and `LessonModel` (id, title, durationSeconds, isLocked, videoPath, transcript).

5.7.6 State Management

`CourseDetailCubit` fetches the course and emits `CourseDetailLoading` | `CourseDetailLoaded` | `CourseDetailError`. `CourseProgressCubit` tracks per-lesson progress locally and syncs it via `POST /courses/{id}/progress`.

5.8 Media Player — Lesson and Book Audio Player

5.8.1 Feature Name

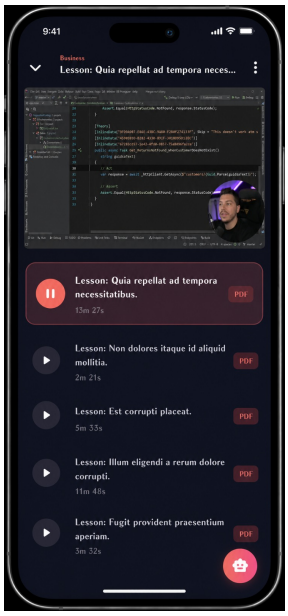
`MediaPlayerScreen` (Video Lessons) and `BookAudioPlayer` (TTS Audiobooks)

5.8.2 Objective

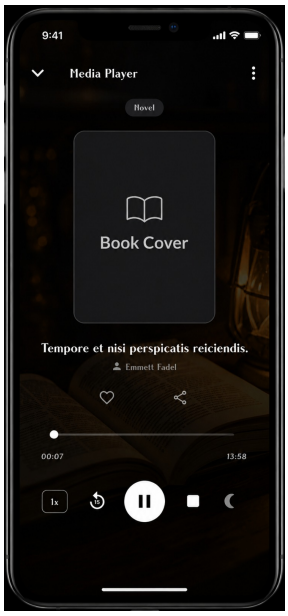
`MediaPlayerScreen` (video lessons) and `BookAudioPlayer` (TTS audiobooks) provide a full-screen media consumption experience for both video lessons and audiobooks, with a persistent Mini Player overlay that continues playback when the user navigates away from the full-screen player.

5.8.3 UI/UX Implementation

- Full-screen video player using `video_player` with custom overlay controls: play/-pause, seek bar, speed selector, fullscreen toggle.
- TTS audio player using `flutter_tts`: chapter list drawer, speed selector (0.5×–2×), progress bar.
- Mini Player — persistent bottom bar visible on all screens (except the full player), showing cover art, title, play/pause, and progress.
- Transcript panel — slides up from the bottom; displays AI-generated text with word-by-word highlight synced to playback.
- AI Chat and Summarise floating action buttons available during lesson playback, grounded on the lesson transcript.



(a) Video Lesson Player with Lesson List and AI Chat Shortcut



(b) Book Audio (TTS) Player with Seek Bar and Speed Control

Figure 5.10. Media Player Screens for Video Lessons and Book Audio

5.8.4 API Endpoints

Table 5.9. Media Player API Endpoints

Method + URL	Description
POST /lessons/{id}/transcribe	Trigger Whisper ASR on server; returns transcript text
POST /chatbot/message	AI chatbot — pass `lesson_id` for transcript-grounded context
POST /books/{id}/chat	Book-specific AI chat (RAG on `pdf_text`)
POST /books/{id}/listen	Log listen event for history & analytics
POST /courses/{id}/progress	Save video watch percentage for continue-watching

5.8.5 Mini Player Controller Architecture

The Mini Player is implemented as a globally accessible overlay registered at the root `MaterialApp`, controlled by a singleton `MiniPlayerController` extending `ChangeNotifier`. Figure 5.11 illustrates how `MiniPlayerController`, `BookDetailCubit`, the `flutter_tts` singleton, and `SharedPreferences` interact.

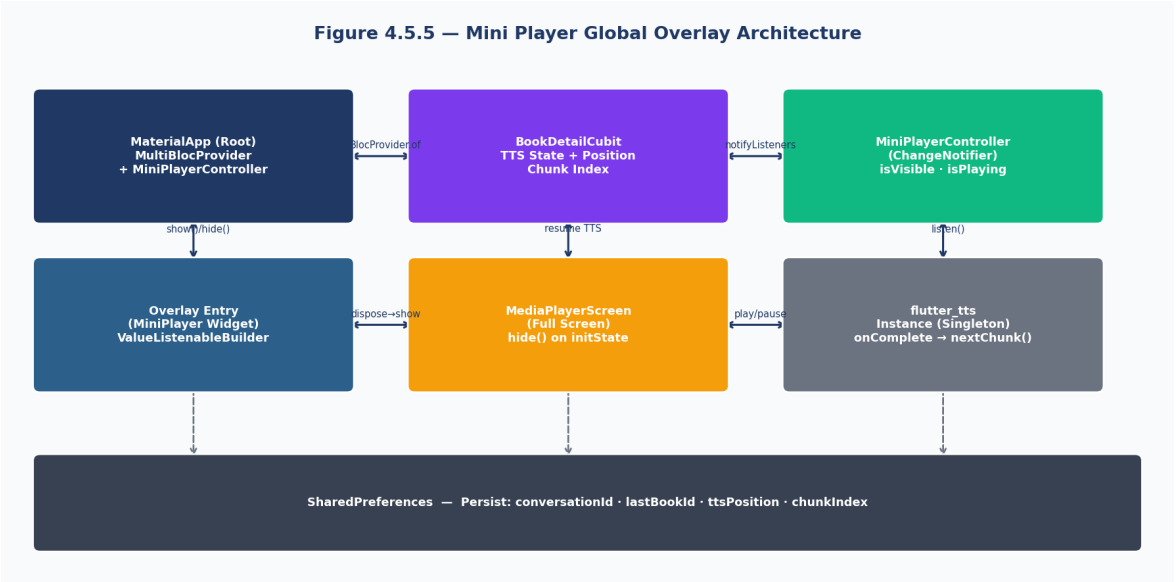


Figure 5.11. Mini Player Global Overlay Architecture

Listing 5.5. MiniPlayerController — Singleton Overlay Controller

```dart
class MiniPlayerController extends ChangeNotifier {
  bool isVisible = false;
  String title = '';
  String coverUrl = '';
  bool isPlaying = false;

  void show({ required String title, required String coverUrl }) {
    isVisible = true;
    this.title = title;
    this.coverUrl = coverUrl;
    notifyListeners();
  }

  void hide() {
    isVisible = false;
    notifyListeners();
  }
}
```

5.8.6 Challenges and Solutions

Table 5.10. Media Player Challenges and Solutions

Challenge	Solution Applied
Mini Player appeared twice on screen	Root-level Overlay entry only; removed local Stack duplicate; added a `mounted` check before `show()`
Mini Player not hiding when entering the full player	Full player `initState()` calls `controller.hide()`; `dispose()` calls `controller.show()` to restore it
TTS stops after `MediaPlayer.pop()`	`MiniPlayerController` holds a reference to `BookDetailCubit`; resumes on `isPlaying` toggle
Large book TTS timeout (> 50 pages)	Text chunked into $\leq$ 4,000-char paragraphs; TTS `onComplete()` chains the next chunk automatically
Video orientation lock on fullscreen	`SystemChrome.setPreferredOrientations()` toggled on enter/exit fullscreen; restored in `dispose()`

5.9 Favourites Management System

5.9.1 Feature Name

`FavoritesScreen` — Books, Authors, Courses, Instructors

5.9.2 Objective

`FavoritesScreen` allows users to save and manage their favourite content across four modules — Books, Authors, Courses, and Instructors — with a unified, globally synchronised, real-time-updated experience.

5.9.3 UI/UX Implementation

- `TabBar` with four tabs: Books, Authors, Courses, Instructors.
- `FavoriteBooksView` — `BookCard` grid layout with heart overlay per card.
- `FavoriteAuthorsView` — `ArtistCard` vertical list.
- `FavoriteCoursesView` — `CourseCard` list with progress indicator.

- `FavoriteInstructorsView` — `ArtistCard` vertical list reused from the Author tab, bound to instructor data.
- Long-press on any item opens a confirmation bottom sheet to remove from favourites.
- Empty state: custom illustration + descriptive message per tab.

Figure 5.12 illustrates how the global `FavoritesBloc` synchronises heart-icon state across multiple screens without redundant API calls.

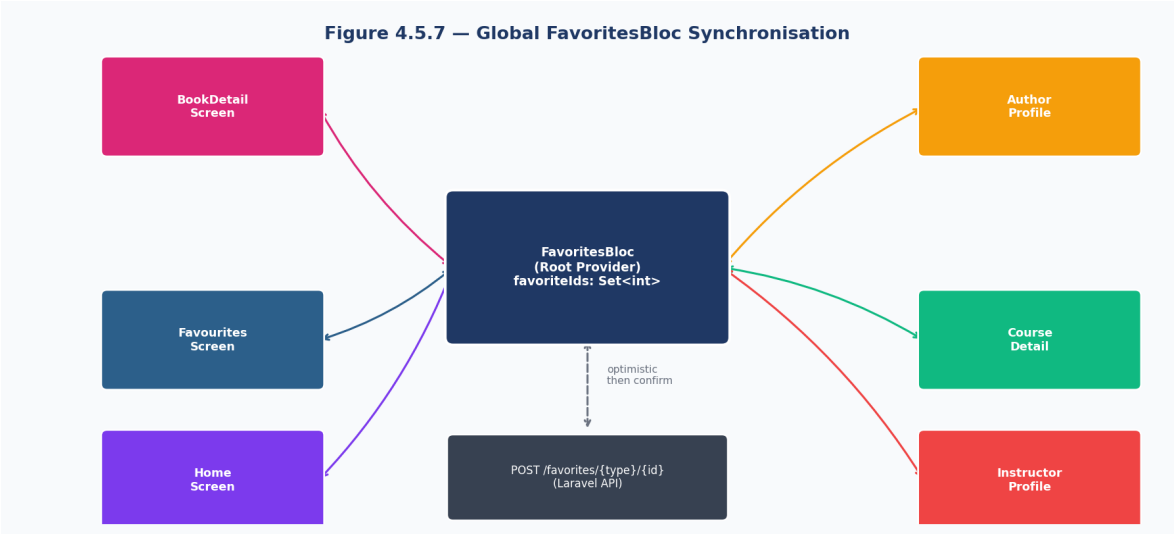


Figure 5.12. Global FavoritesBloc Synchronisation Diagram

5.9.4 API Endpoints

Table 5.11. FavoritesScreen API Endpoints

Method + URL	Description
GET /favorites/{type}	type = books | authors | courses | instructors
POST /favorites/book/{id}	Toggle book favourite
POST /favorites/author/{id}	Toggle author favourite (follow)
POST /favorites/course/{id}	Toggle course favourite
POST /favorites/instructor/{id}	Toggle instructor follow

5.9.5 Sample Response — GET /favorites/books (200 OK)

Listing 5.6. GET /favorites/books — Sample Response

```
{ "data": [
    { "id":12, "title":"Clean Code", "cover_image":"...",
      "rating":4.7,
      "author": { "id":3, "name":"Robert C. Martin" } }
```

```
4   ] }
```

5.9.6 FavoritesBloc State Design

- Events: `LoadFavorites(type)` | `ToggleFavorite(type, id)` | `RemoveFavorite(type, id)`.
- States: `FavoritesLoading` | `FavoritesLoaded(favoriteIds: Set<int>)` | `FavoritesError`
- The loaded state stores a `Set<int>` of favourite IDs; any widget checks `isFav = state.favoriteIds.contains(id)`.
- Optimistic update: ID added/removed from the set immediately; an API error triggers a revert and a SnackBar notification.

5.9.7 Integration Steps

1. `FavoritesBloc` is provided at the root `MaterialApp` via `MultiBlocProvider` — a single instance app-wide.
2. Each screen dispatches `LoadFavorites` on first build (cached if already loaded).
3. The heart icon reads `BlocSelector<FavoritesBloc, FavoritesState, bool>` — no full rebuild.
4. A tap dispatches `ToggleFavorite` → the bloc calls the API → emits an updated `FavoritesLoaded` state.

5.10 AI Chatbot — General and Contextual

5.10.1 Objective

The AI chatbot provides an always-accessible, context-aware conversational AI assistant. In book screens it is grounded on the book's extracted PDF text; in lesson screens it is grounded on the Whisper-generated transcript, enabling RAG-based, accurate responses.

5.10.2 UI/UX Implementation

- Full-screen `ChatScreen`: scrollable bubble list, text input bar, "Summarise" shortcut floating action button.
- User bubbles — right-aligned, blue background.
- AI bubbles — left-aligned, grey background, content rendered with `flutter_markdown`.
- Typing indicator (three animated dots) while awaiting the API response.

- Conversation history list accessible from the AppBar — switch or delete past conversations.

Figure 5.13 shows the complete AI Chat & Summarise request/response flow including RAG context injection, and Figure 5.14 shows the three chatbot surfaces in the app: the general support assistant, the book-grounded chat, and the AI summary bottom sheet.

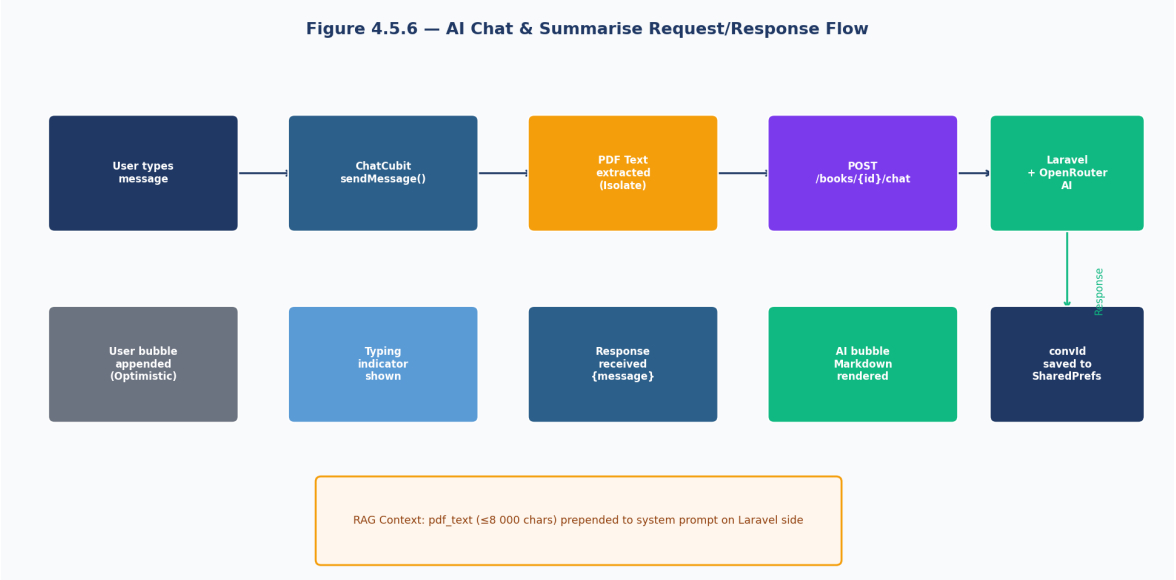
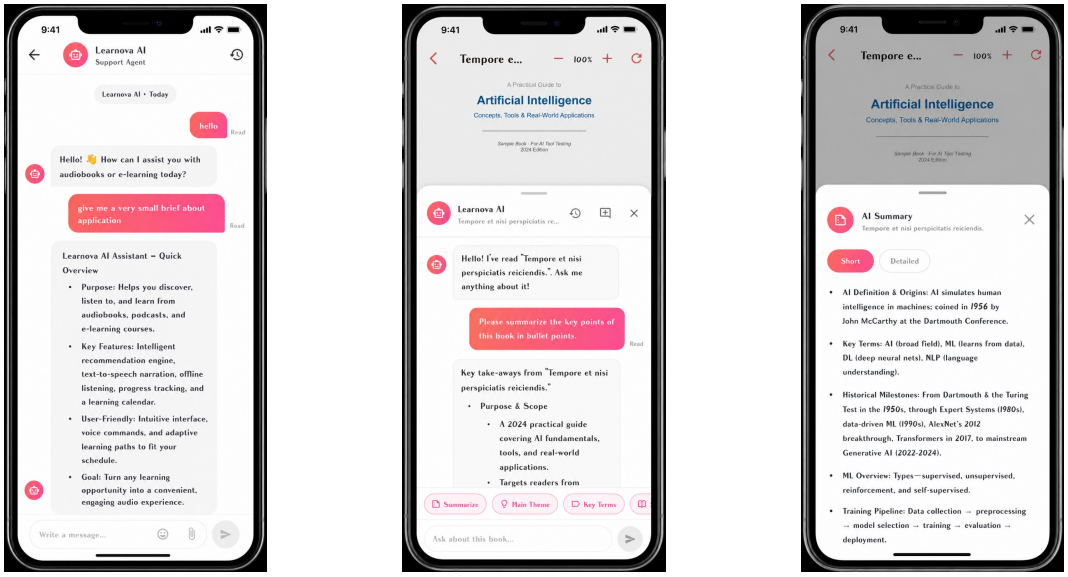


Figure 4.5.6 — AI Chat & Summarise Request/Response Flow
User types message
ChatCubit sendMessage()
PDF Text extracted (Isolate)
POST /books/{id}/chat
Laravel + OpenRouter AI
Response
User bubble appended (Optimistic)
Typing indicator shown
Response received {message}
AI bubble Markdown rendered
convId saved to SharedPrefs
RAG Context: pdf_text (≤8 000 chars) prepended to system prompt on Laravel side

Figure 5.13. AI Chat & Summarise Request/Response Flow


9:41
Learnova AI
Support Agent
hello
Hello! 👋 How can I assist you with audiobooks or e-learning today?
give me a very small brief about application
Learnova AI Assistant – Quick Overview
Purpose: Helps you discover, listen to, and learn from audiobooks, podcasts, and e-learning courses.
Key Features: Intelligent recommendation engine, text-to-speech narration, offline listening, progress tracking, and a learning calendar.
User-Friendly: Intuitive interface, voice commands, and adaptive learning paths to fit your schedule.
Goal: Turn any learning opportunity into a convenient, engaging audio experience.
Write a message...
Tempore e...
Artificial Intelligence
Concepts, Tools & Real-World Applications
Learnova AI
Hello! I've read "Tempore et nisi perspiciatis reiciendis." Ask me anything about it!
Please summarize the key points of this book in bullet points.
Key take-aways from "Tempore et nisi perspiciatis reiciendis."
Purpose & Scope
A 2024 practical guide covering AI fundamentals, tools, and real-world applications.
Targets readers from
Summarize
Main Theme
Key Terms
Ask about this book...
AI Summary
Short
Detailed
AI Definition & Origins: AI simulates human intelligence in machines; coined in 1956 by John McCarthy at the Dartmouth Conference.
Key Terms: AI (broad field), ML (learns from data), DL (deep neural nets), NLP (language understanding).
Historical Milestones: From Dartmouth & the Turing Test in the 1950s, through Expert Systems (1980s), data-driven ML (1990s), AlexNet's 2012 breakthrough, Transformers in 2017, to mainstream Generative AI (2022-2024).
ML Overview: Types—supervised, unsupervised, reinforcement, and self-supervised.
Training Pipeline: Data collection → preprocessing → model selection → training → evaluation → deployment.

(a) General Support Chatbot **(b)** Book-Grounded Chat over PDF **(c)** AI Summary — Short / Detailed

Figure 5.14. AI Chatbot Surfaces: General Support, Contextual Book Chat, and AI Summary

5.10.3 API Endpoints

Table 5.12. AI Chatbot API Endpoints

Method + URL	Description
POST /chatbot/message	General chatbot (pass `lesson_id` for transcript context)
POST /books/{id}/chat	Book-specific chat (pass `pdf_text` for RAG context)
GET /chat/conversations	List all conversations for the authenticated user
GET /chat/conversations/{id}	Load a conversation with full message history
DELETE /chat/conversations/{id}	Delete a specific conversation

Listing 5.7. POST /chatbot/message — Request and Response

```
// Request
{ "message": "Explain polymorphism", "history":
   [{"role":"user","content":"..."}],
  "conversation_id": 7, "lesson_id": 42 }
// Response
{ "id": "xyz789", "message": "Polymorphism allows objects...",
  "role": "assistant", "conversation_id": 7 }
```

5.10.4 State Management

`ChatCubit` holds the list of `ChatMessageModel`, the `conversationId`, and an `isLoading` flag. Its `sendMessage(text)` method appends a user bubble optimistically, calls the API, and appends the AI reply on success or an error bubble on failure.

5.10.5 Challenges and Solutions

Table 5.13. AI Chatbot Challenges and Solutions

Challenge	Solution Applied
Context lost between app restarts	`conversation_id` persisted in `SharedPreferences`; reloaded via `GET /chat/conversations/{id}` on screen open
Large PDF text causing 413 Payload Too Large	`pdf_text` truncated to the first 8,000 characters client-side before sending
Markdown rendering breaking chat bubble width	`MarkdownBody` wrapped in a `ConstrainedBox` with `maxWidth = 0.75 * screenWidth`

5.11 BLoC / Cubit State Machine

All Cubits and BLoCs in Learnova follow a consistent three-state pattern (Loading → Loaded / Error) with optimistic intermediate states for write operations. Figure 5.15 illustrates this pattern using `BookDetailCubit` as a representative example.

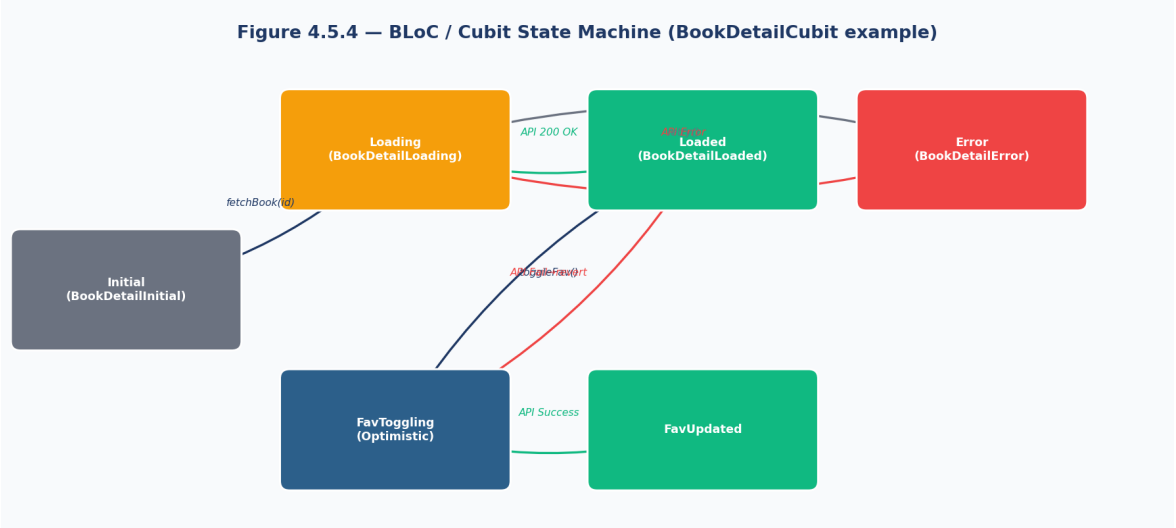


Figure 5.15. BLoC/Cubit State Machine — BookDetailCubit Example

Figure 5.16 further illustrates the general event/state flow pattern: the UI layer emits Events, the BLoC processes them and emits States back to the UI, while data requests and responses flow through the Repository layer.

Figure 5.16. BLoC Event/State Flow: UI ↔ BLoC ↔ Data Layer

The same pattern is applied uniformly across `BookDetailCubit`, `CourseDetailCubit`, `ChatCubit`, `SearchCubit`, `HomeCubit`, and every other Cubit in the application, ensuring predictable and testable state transitions throughout.

5.12 REST API Endpoint Map

The Flutter client consumes API endpoint groups organised by domain — authentication, books, courses, favourites, search, and AI services. Figure 5.17 provides a visual overview of all API endpoint groups consumed by the Flutter client. The central Flutter Client node communicates bidirectionally with each API group via Dio interceptors carrying the Sanctum Bearer token.

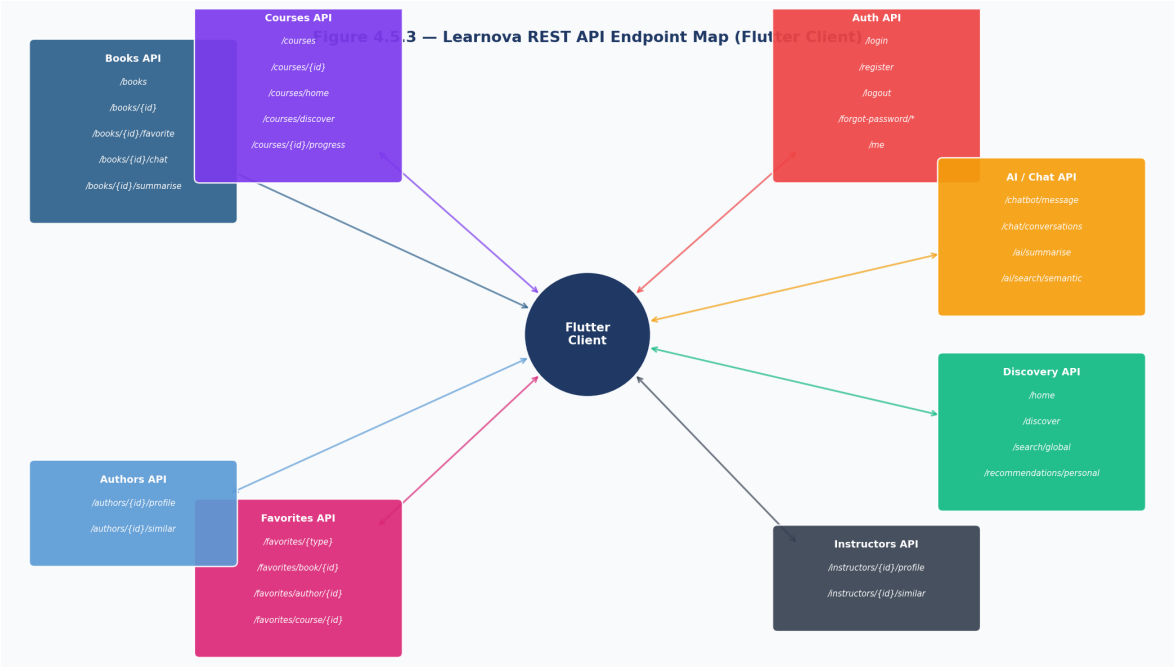


Figure 5.17. Learnova REST API Endpoint Map (Flutter Client)

5.13 Navigation and Routing (go_router)

All routes are declared in a centralised `AppRouter` class. An authentication guard redirects unauthenticated deep-links to `/login`. Figure 5.18 shows the complete routing tree.

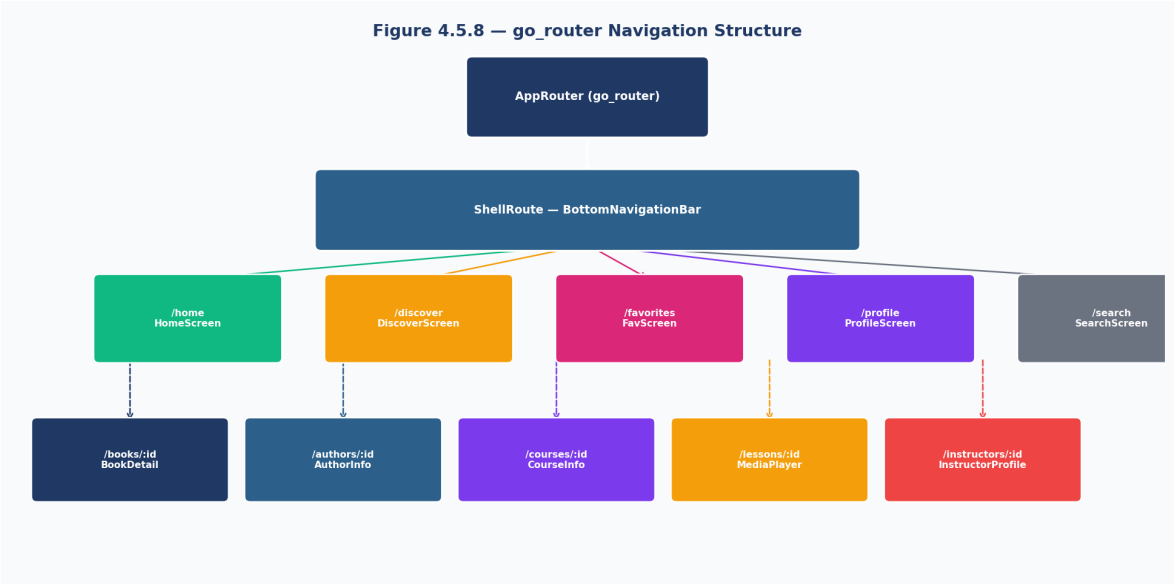


Figure 5.18. go_router Navigation Structure

Table 5.14. go_router Route Table

Route Path	Screen
/	SplashScreen
/onboarding	OnboardingScreen
/login	LoginScreen
/register	RegisterScreen
/home (ShellRoute)	HomeScreen with BottomNavigationBar
/books/:id	BookDetailScreen
/books/:id/player	BookAudioPlayer
/authors/:id	AuthorInfoScreen
/courses/:id	CourseInfoScreen
/lessons/:id	MediaPlayerScreen
/instructors/:id	InstructorProfileScreen
/favorites	FavoritesScreen
/search	SearchScreen
/chat	ChatScreen

5.13.1 Key Routing Decisions

- `ShellRoute` wraps Home, Discover, Favourites, and Profile tabs with a persistent `BottomNavigationBar`.
- All entity IDs are passed as `String` path parameters; converted to `int` only inside the repository layer.
- Deep-link support: `learnova://books/12` resolves to `BookDetailScreen(id: 12)` via a `go_router` redirect.

5.14 Error Handling and UX Patterns

A consistent error-handling and UX state pattern is applied to all API-backed screens in the application, as illustrated in Figure 5.19.

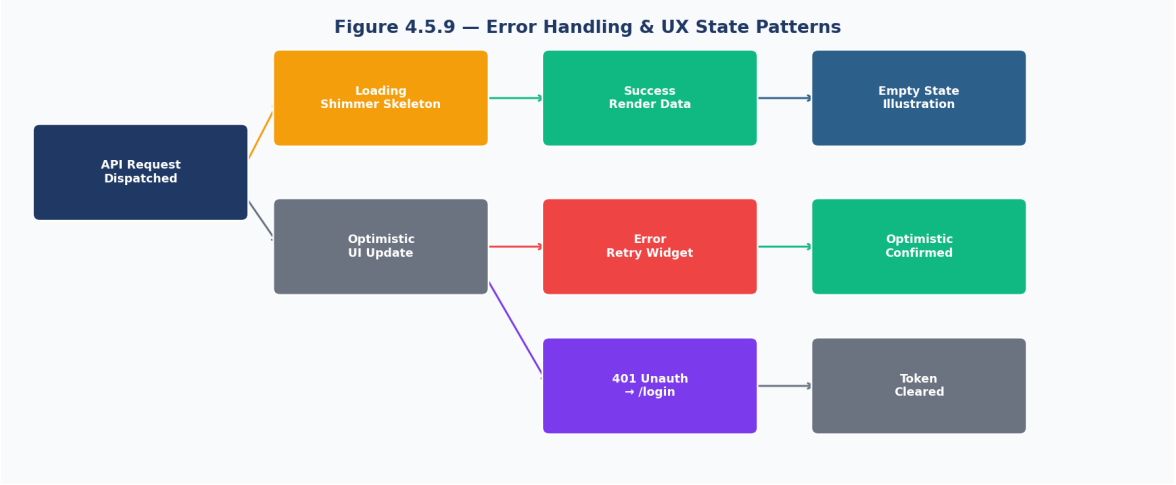


Figure 5.19. Error Handling & UX State Patterns

Table 5.15. Error Handling and UX State Patterns

Pattern	**Implementation**
Loading state	Shimmer skeleton widgets matching the screen layout (`shimmer` package)
Network error	Full-screen ErrorWidget with retry button; `DioException` caught at the repository layer
Empty state	Custom illustration + descriptive message per screen
Optimistic updates	State updated before the API response; reverted on error with a SnackBar notification
Auth expiry (401)	Dio interceptor catches 401 → clears SharedPrefs token → redirects to `/login`
Input validation	Form fields validated client-side before the API call (required, email, min length)
Offline detection	`connectivity_plus` monitors network; banner shown when offline

5.15 Overall Challenges and Solutions

Table 5.16. Overall Flutter Implementation Challenges and Solutions

Challenge	Solution Applied
Multiple similar profile screens duplicating UI code	Shared `GradientProfileHeader` widget parameterised with `imageUrl`, `title`, `subtitle`, `actionButton`
ID type inconsistency across the API (int vs String)	Centralised normalisation: all IDs parsed as String in models, cast to int only when required by the API
Global FavoritesBloc synchronisation	Single `BlocProvider` at root; all screens dispatch to the same instance; no duplicate API calls
Performance with long horizontal lists	`ListView.builder` with `cacheExtent`; `cached_network_image` with fade-in transition
Complex gradient header over the status bar	`Stack` + `SafeArea` + `extendBodyBehindAppBar:true`; gradient starts from `top=0`
API call storms on rapid navigation	Repository caches the last successful response for 60s using an in-memory map keyed by endpoint + params
Flutter TTS competing with the system media session	`AudioSession.instance.configure(AudioSessionCo` called on app launch
Git merge conflicts during parallel feature branches	Feature branch per screen; rebase-before-merge policy; shared widget folder locked to one owner

Chapter Summary

This chapter documented the complete Flutter mobile implementation of Learnova's core features, covering every screen, API endpoint, state management strategy, and integration detail required for the mobile client to function as a production-ready application. Key achievements include a rich `BookDetailScreen` combining PDF viewing, TTS audio playback, AI summarisation, and contextual chat in one tabbed interface; full instructor and author profile screens with optimistic follow/unfollow interactions and consistent reusable header widgets; a globally synchronised `FavoritesBloc` ensuring consistent heart-icon state across all screens without redundant API calls; a per-

sistent Mini Player architecture that survives navigation events and correctly resumes TTS playback after the full-screen player is dismissed; a context-aware AI chatbot grounded either on the current book's extracted PDF text or the lesson's Whisper-generated transcript; and robust error handling with optimistic UI patterns, shimmer loading skeletons, and automatic token-expiry redirection. The integration between the Flutter client and the Laravel REST API was achieved entirely through the Dio HTTP client with a Sanctum Bearer token interceptor, ensuring all authenticated endpoints are accessed securely and consistently throughout the application.

Chapter 6

System Testing: Backend, AI Microservices & Flutter

Contents

6.1 Overview 101
6.2 Testing Objectives 101
6.3 Testing Strategy and Levels 101
6.4 Test Environment and Tools 102
6.5 Backend (Laravel) Testing 103
6.6 AI Microservices Testing 106
6.7 Flutter Mobile Testing 107
6.8 Integration Testing 108
6.9 Performance Testing 109
6.10 Security Testing 110
6.11 User Acceptance Testing (UAT) 111
6.12 Test Results Summary 112
6.13 Challenges Faced 112
6.14 Future Testing Enhancements 113

6.1 Overview

Testing is a critical phase in the software development lifecycle, ensuring that Learnova functions correctly, reliably, and securely across its three major components: the Flutter mobile application, the Laravel RESTful API, and the six Python AI microservices. Because Learnova integrates multiple heterogeneous technologies communicating over HTTP, Kafka, and Redis, a disciplined, multi-layered testing strategy was essential to catch defects early, verify correct behaviour under real-world conditions, and validate that the system meets its functional and non-functional requirements before the final defense.

This chapter documents the testing methodology adopted throughout the project, the tools used at each layer, representative test cases and their outcomes, and the key challenges encountered while testing a distributed, AI-integrated system.

6.2 Testing Objectives

- Verify that all functional requirements are correctly implemented across the Flutter, Laravel, and AI layers.
- Ensure the correctness of the RESTful API, including proper HTTP status codes, response payloads, and validation error handling.
- Confirm that all six AI microservices return accurate, well-formed responses and degrade gracefully when unavailable.
- Validate the end-to-end integration flow: Flutter → Laravel → AI microservices → Redis/Pinecone → back to the mobile client.
- Assess system performance under realistic load, particularly for latency-sensitive features such as semantic search and RAG streaming.
- Identify and mitigate security vulnerabilities in authentication, authorization, and data handling.
- Confirm that the mobile application delivers a smooth, error-free user experience across different network conditions and device states.

6.3 Testing Strategy and Levels

A layered testing strategy was adopted, moving from isolated unit-level verification to full end-to-end system validation. Table 6.1 summarizes the testing levels applied and their respective scope.

Table 6.1. Testing Levels Applied Across Learnova

Level	**Scope**	**Primary Owner**
Unit Testing	Individual functions, model methods, service classes, widgets in isolation	Each development team
Integration Testing	Interaction between Laravel and AI microservices; Flutter and Laravel API	Backend & AI teams
System Testing	Full user flows across the mobile app end-to-end	Whole team
Performance Testing	Response time and throughput under concurrent load	Backend & AI teams
Security Testing	AuthN/AuthZ, input validation, token handling	Backend team
User Acceptance Testing	Real device walkthroughs against requirements	Whole team + supervisor feedback

This bottom-up approach ensured that defects were caught as early as possible — at the unit level, before they could propagate into more expensive-to-fix integration or system-level failures.

6.4 Test Environment and Tools

Testing was conducted primarily on a local development environment using XAMPP for the Laravel/MySQL stack, Docker Compose for the six AI microservices, and physical Android devices connected over the local network for Flutter testing.

Table 6.2. Testing Tools by Layer

Layer	Tool	Purpose
Backend (Laravel)	PHPUnit	Automated unit and feature tests
Backend (Laravel)	Postman	Manual API testing and regression checks
AI Microservices	pytest	Unit tests for chunking, encoding, retrieval logic
AI Microservices	Postman / curl	Manual endpoint verification and /health checks
Flutter	flutter_test	Widget and unit tests for Cubits/BLoCs
Flutter	Manual QA	Real-device walkthroughs on Android
Database	MySQL Workbench	Schema and data integrity verification
Infrastructure	Docker Health Checks	Verify all AI microservice containers via /health

Table 6.3. Test Environment Configuration

Component	Configuration
Backend Server	XAMPP (Apache + PHP 8.1), Laravel 12, localhost:8000
Database	MySQL 8.0 via XAMPP, local schema seeded with factories
AI Services	Docker Compose, 6 containers (ports 8000–8004 + embedding consumer)
Mobile Devices	Physical Android devices on the same LAN, connected via local IP
Network	Windows Firewall rules opened for LAN-based mobile testing

6.5 Backend (Laravel) Testing

6.5.1 Unit and Feature Testing with PHPUnit

PHPUnit was used to write automated feature tests covering authentication, CRUD operations, and validation rules. Each test seeds a fresh database state using Laravel's `RefreshDatabase` trait and factories, ensuring tests remain isolated and repeatable.

Listing 6.1. PHPUnit Feature Test for Book Favouriting

```
public function test_authenticated_user_can_toggle_favorite_book()
{
    $user = User::factory()->create();
    $book = Book::factory()->create();
```

```
5
6       $response = $this->actingAs($user, 'sanctum')
7           ->postJson("/api/favorites/book/{$book->id}");
8
9       $response->assertStatus(200)
10          ->assertJsonStructure(['is_favorite']);
11
12      $this->assertDatabaseHas('bookmarks', [
13          'user_id' => $user->id,
14          'book_id' => $book->id,
15      ]);
16  }
```

6.5.2 Manual API Testing with Postman

A comprehensive Postman collection was maintained covering all endpoint groups: authentication, books, courses, favourites, search, and the AI proxy routes. Each endpoint was manually verified for correct status codes, response schema, and edge-case handling such as missing parameters, invalid IDs, and unauthorized access.

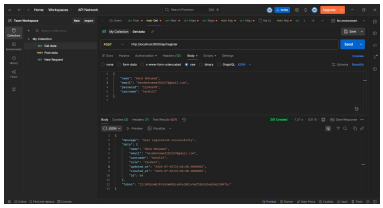
Table 6.4. Sample Backend Test Cases and Results

Endpoint	Method	Test Case	Result
/api/register	POST	Valid registration payload	Pass — 201 Created
/api/register	POST	Duplicate email address	Pass — 422 Validation Error
/api/login	POST	Valid credentials	Pass — 200 OK + Sanctum token
/api/login	POST	Incorrect password	Pass — 401 Unauthorized
/api/topics/save	POST	Payload sent as 'topics' instead of 'category_ids'	Fail (initial) — 422 key mismatch; fixed
/api/topics/save	POST	Names sent instead of numeric IDs	Fail (initial) — FormatException; fixed
/api/courses/{id}/videos	POST	Multipart upload with video + PDF	Pass — 201 Created
/api/favorites/book/{id}	POST	Toggle favourite without auth token	Pass — 401 Unauthorized
/api/ai/health	GET	All AI containers running	Pass — 200 OK, all healthy

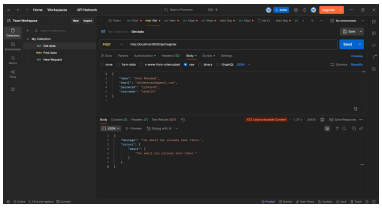
Note
The two failing cases in the topics onboarding flow (key mismatch and name-vs-ID mismatch) were identified during manual Postman testing before being resolved in the Flutter data layer, illustrating the value of testing the full request/response contract rather than relying on backend assumptions alone.

6.5.3 Postman Test Evidence

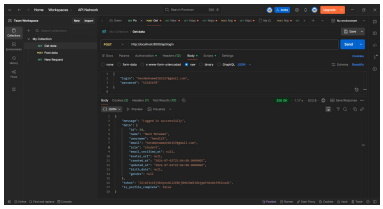
Figures 6.1 through 6.3 present representative Postman screenshots captured during backend testing, corresponding to the test cases summarised in Table 6.4.



(a) POST /api/register — 201 Created

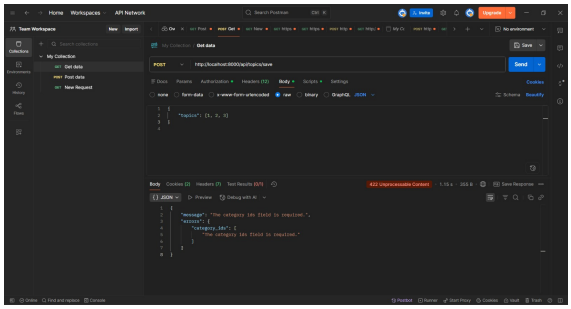


(b) POST /api/register — Duplicate Email (422)

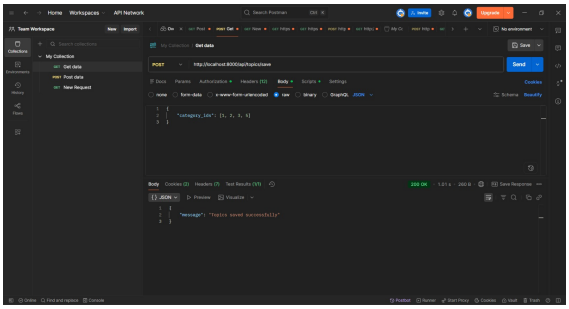


(c) POST /api/login — 200 OK + Sanctum Token

Figure 6.1. Postman Evidence — Registration and Login Endpoints

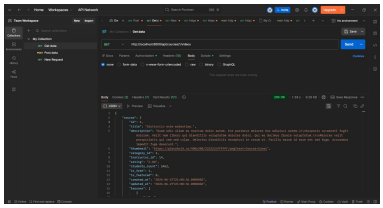


(a) POST /api/topics/save — Initial Payload (`topics` key) Rejected with 422

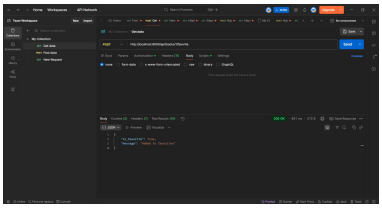


(b) POST /api/topics/save — Corrected Payload (`category_ids` key) — 200 OK

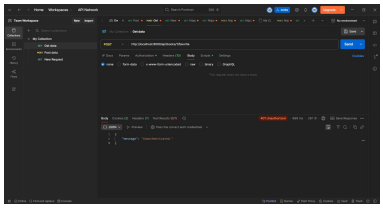
Figure 6.2. Postman Evidence — The Topics Onboarding Key-Mismatch Fix



(a) GET /api/courses/1/videos — 200 OK



(b) POST /api/books/1/favorite — 200 OK



(c) POST /api/books/1/favorite — No Token (401)

Figure 6.3. Postman Evidence — Course Content and Favourite Toggle Endpoints

6.6 AI Microservices Testing

Each of the six Python AI microservices was tested independently before integration, using a combination of pytest unit tests and manual endpoint verification through Postman and curl.

Table 6.5. AI Microservice Test Coverage

Service	Unit-Level Focus	Integration-Level Focus
Ingestion	Text extraction correctness, chunk size/overlap boundaries, blank-page filtering	Kafka publish confirmation, status transitions (queued → processing → done)
Embedding	Vector dimensionality (768-d), L2 normalisation correctness	Batch flush at 100 records, DLQ routing on repeated upsert failure
Search	Query encoding correctness, cache key hashing (SHA-256)	End-to-end ANN retrieval (top 50) + cross-encoder re-ranking (top 10)
Summarisation	Prompt formatting per summary type, beam search output shape	Map-reduce pipeline correctness on multi-batch books; Redis TTL caching
RAG Q&A	Similarity threshold filtering (0.30 cut-off), token budget truncation	SSE streaming event sequence, 15s inter-token timeout behaviour
Recommendation	NCF forward pass output range [0,1], alpha selection by user tier	Hybrid score fusion, cold-start endpoint for new users

6.6.1 Example Test Case — RAG Hallucination Guard

A dedicated test verified that the RAG service correctly refuses to answer when no retrieved passage exceeds the 0.30 similarity threshold, returning the standardised insufficient-information message rather than a fabricated answer.

Listing 6.2. Test for the RAG Hallucination Guard

```
def
    test_rag_returns_fallback_when_no_passage_meets_threshold(mock_retriever):
    mock_retriever.retrieve.return_value = []  # simulate no
        passages >= 0.30

    response = rag_chain.invoke({
        "question": "What is the capital of Mars?",
        "isbn": "9780132350884",
    })

    assert "I don't have enough information" in response["answer"]
    assert response["citations"] == []
```

6.6.2 AI Service Health Checks

All six services expose a `/health` endpoint polled by Laravel's `/api/ai/health` aggregation route. During testing, each service was deliberately stopped one at a time to confirm that Laravel's `AiService` class correctly returns a graceful, descriptive error rather than a raw connection exception or a crashed request.

6.7 Flutter Mobile Testing

6.7.1 Widget and Cubit Testing

The `flutter_test` package was used to verify Cubit state transitions in isolation, ensuring that Loading → Loaded/Error transitions behave predictably for key Cubits such as `BookDetailCubit`, `FavoritesBloc`, and `ChatCubit`.

Listing 6.3. Cubit Test for FavoritesBloc

```
testWidgets('toggling favorite updates state optimistically',
    (tester) async {
  final bloc = FavoritesBloc(repository: mockRepository);

  bloc.add(ToggleFavorite('book', 12));

  await expectLater(
    bloc.stream,
    emitsInOrder([
      isA<FavoritesLoaded>().having(
        (s) => s.favoriteIds.contains(12), 'contains id', true,
      ),
    ]),
  );
});
```

6.7.2 Manual QA on Physical Devices

Because Learnova relies heavily on native device features — text-to-speech, video playback, PDF rendering, and persistent overlays — manual testing on physical Android devices over the local network was essential to catch issues not reproducible on emulators alone.

Table 6.6. Sample Flutter Manual Test Cases

Test Case	**Expected Behaviour**	**Result**
Play TTS audio for a 120-page book	Text splits into ≤4,000-char chunks; playback auto-advances without gaps	Pass (after chunking fix)
Navigate away during TTS playback	Mini Player appears and playback continues uninterrupted	Pass
Enter full-screen video player	Mini Player hides; orientation locks to landscape	Pass
Exit full-screen video player	Mini Player reappears; orientation restores to portrait	Pass
Toggle book favourite offline	Optimistic UI update, then reverts with SnackBar on failure	Pass
Complete onboarding topic selection	POST /topics/save succeeds with correct category IDs	Pass (after key-mismatch fix)
Access app via LAN IP after switching Wi-Fi networks	Connection fails gracefully with a retry prompt	Pass (after firewall fix)
Ask a question in Book Chat with a 15,000-char PDF	Client truncates pdf_text to 8,000 characters; no 413 error	Pass

6.8 Integration Testing

Integration testing focused on validating the complete request path across all three layers, particularly for the AI proxy pattern used throughout the backend. Table 6.7 summarizes the key end-to-end flows verified.

Table 6.7. Verified End-to-End Integration Flows

Flow	Path	Result
Semantic search	Flutter → Laravel /api/ai/search/semantic → search_service:8002 → Redis cache check → Pinecone ANN + re-rank → response	Pass
Book ingestion	Admin upload → ingestion:8000 → Kafka book.chunks → embedding consumer → Pinecone upsert	Pass (validated via direct_ingest.py bypass)
Bulk ingestion of catalogue	bulk_ingest.py → sequential ingestion of 101 books → Pinecone namespaces per ISBN	Pass
RAG Q&A streaming	Flutter chat screen → Laravel /api/ai/qa/stream → rag_service:8004 SSE → token-by-token render in ChatCubit	Pass
Recommendations on Home screen	Flutter Home → Laravel /api/home → rec_service:8001 hybrid NCF+CBF → ranked book list	Pass
Lesson transcription	Admin triggers transcribe:all-lessons → Whisper processing → transcript stored → lesson chat grounded	Pass

Note

During early integration testing, the semantic search chain occasionally stalled due to Kafka consumer lag when ingesting the full 101-book catalogue. A custom `direct_ingest.py` script was used to bypass Kafka for targeted re-testing, allowing the search integration to be validated independently of the ingestion pipeline's throughput.

6.9 Performance Testing

Basic performance testing was conducted to ensure that latency-sensitive endpoints remain responsive under typical classroom-scale concurrent usage.

Table 6.8. Observed Response Times for Key Endpoints

Endpoint	Avg. Response Time	Notes
/api/ai/search/semantic	∼250–400ms	Includes ANN retrieval + re-ranking; <50ms on Redis cache hit
/api/ai/recommendations	∼150–300ms	Redis-cached for 5 minutes per user
/api/ai/summarise	3–8s (first request)	Map-reduce over chunk batches; instant on 24h cache hit
/api/ai/qa/stream (first token)	∼800ms–1.2s	Time to first SSE token from Groq Llama-3.3-70b
/api/books/{id}	<100ms	Simple Eloquent query with eager-loaded relations

The Recommendation Service's Redis sliding-window rate limiter (60 requests per minute per user) was tested by issuing rapid successive requests, which correctly returned an HTTP 429 response once the limit was exceeded.

6.10 Security Testing

Security-focused test cases verified that authentication and authorization boundaries were correctly enforced across the API surface.

Table 6.9. Security Test Cases

Test Case	Expected Behaviour	Result
Access protected endpoint without a Sanctum token	401 Unauthorized returned	Pass
Access admin-only endpoint as a student role	403 Forbidden returned	Pass
Submit SQL-injection-style input in search query	Parameterised via Eloquent; no injection possible	Pass
Reuse an OTP code after successful password reset	422 Validation Error — OTP already consumed	Pass
Send an expired Sanctum token	401 Unauthorized; token cleared, redirect to login	Pass
Upload a non-PDF/EPUB file to the ingestion endpoint	422 Validation Error — unsupported file type	Pass

6.11 User Acceptance Testing (UAT)

User Acceptance Testing was carried out internally by the six-member project team, each walking through complete user journeys on physical Android devices and comparing the observed behaviour against the project objectives. Feedback from the project supervisor, Dr. Sayed Noah, during progress reviews was incorporated as an informal acceptance checkpoint ahead of the formal defense.

Table 6.10. UAT Scenarios and Outcomes

Scenario	**Acceptance Criteria**	**Outcome**
New user completes onboarding and receives a personalized home feed	Recommendations reflect selected topics within the first session	Accepted
User searches using a natural-language query rather than exact title	Relevant results returned even without exact keyword match	Accepted
User asks a question about a book's content	Answer is grounded in the book text with a page citation	Accepted
User listens to a book via TTS while browsing other screens	Playback continues via Mini Player without interruption	Accepted
User completes a course video and revisits later	Progress correctly resumes from the last watched position	Accepted

6.12 Test Results Summary

Table 6.11. Overall Testing Summary

Layer	Test Cases Executed	Pass Rate	Notable Fixes
Backend (Laravel)	60+ (PHPUnit + Postman)	High, after fixes	Topics onboarding key mismatch; MySQL/InnoDB corruption recovery
AI Microservices	40+ (pytest + Postman)	High	Kafka consumer lag during bulk ingestion, mitigated with direct_ingest.py
Flutter Mobile	30+ (widget + manual)	High, after fixes	TTS timeout on long books; Mini Player duplication; LAN/firewall issues
Integration	6 major end-to-end flows	All passing	Full chain validated from Flutter through Laravel to AI services and back

Summary
Overall, the testing process surfaced a set of concrete, reproducible defects early in development — most notably around request payload contracts (the 'topics' vs 'category_ids' mismatch), asynchronous pipeline throughput (Kafka lag during bulk ingestion), and native mobile behaviour (TTS chunking and Mini Player lifecycle) — all of which were resolved and re-verified before the final defense.

6.13 Challenges Faced

- **Testing Across a Distributed Stack:** Coordinating tests across Flutter, Laravel, and six independently deployed Python microservices required careful sequencing — verifying each service in isolation before attempting full end-to-end integration tests.
- **Kafka Consumer Lag During Bulk Testing:** Testing the full 101-book ingestion pipeline exposed consumer lag in the Embedding Service under bulk load. The `direct_ingest.py` bypass script was created to isolate and re-test the Search Service independently of ingestion throughput.

- **Local Network Testing Constraints:** Testing the Flutter app on physical devices required resolving localhost vs LAN IP mismatches and configuring Windows Firewall rules to allow inbound connections to the local Laravel server.
- **Non-Deterministic LLM Outputs:** Because the RAG and Summarisation services rely on generative models, exact output text varies between runs. Test assertions were designed around structural properties (citation presence, response format, threshold behaviour) rather than exact text matches.
- **Database State Corruption During Repeated Test Runs:** Repeated local testing occasionally triggered MySQL/InnoDB corruption in the XAMPP environment, requiring table repair and, in some cases, a full local database reseed via Laravel factories and seeders.
- **Limited Automated Coverage for AI Services:** Given the project timeline, automated pytest coverage for the AI microservices was prioritized around the most safety-critical logic (the RAG similarity threshold and hallucination guard), with broader coverage handled through manual Postman verification.

6.14 Future Testing Enhancements

- Introduce continuous integration (CI) pipelines to run PHPUnit, pytest, and Flutter tests automatically on every push.
- Expand automated widget test coverage to all Cubits and BLoCs, not only the highest-risk ones.
- Add load-testing tools (e.g. k6 or Locust) to formally benchmark AI microservice throughput under sustained concurrent load.
- Introduce contract testing between Flutter and Laravel to catch request/response schema mismatches automatically.
- Add end-to-end UI testing using `integration_test` to automate the manual device walkthroughs currently performed for UAT.

Chapter Summary

This chapter documented the comprehensive, multi-layered testing strategy applied to Learnova, spanning unit, integration, system, performance, security, and user acceptance testing across the Flutter mobile application, the Laravel backend, and the six Python AI microservices. Automated tools — PHPUnit, pytest, and flutter_test — were combined with extensive manual verification through Postman and real-device QA to validate both individual components and complete end-to-end flows such as

semantic search, book ingestion, and RAG-based question answering. Key defects, including a request-payload key mismatch in the onboarding flow, Kafka consumer lag during bulk ingestion, and native TTS/Mini Player lifecycle issues, were identified and resolved prior to the final defense. The resulting test coverage provides confidence that Learnova's core features function correctly, degrade gracefully under partial failure, and deliver an acceptable user experience across realistic usage scenarios.

Bibliography

[1] R. Tiwari, "The integration of AI and machine learning in education and its potential to personalize and improve student learning experiences," *International Journal of Scientific Research in Engineering and Management*, vol. 7, no. 2, 2023.

[2] Y. Jayaram, "AI-driven personalization 2.0: Hyper-personalized journeys for every student type," *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, 2024.

[3] F. L. da Silva *et al.*, "A systematic literature review on educational recommender systems," *Education and Information Technologies*, 2023.

[4] V. Authors, "Recommender systems: A comprehensive review (2017–2024)," *arXiv preprint*, 2024.

[5] N. Yanes *et al.*, "A machine learning-based recommender system for improving students learning experiences," *IEEE Access*, vol. 8, pp. 201218–201235, 2020.

[6] Q. EL Maazouzi, A. Retbi, and S. Bennani, "Optimizing recommendation systems in E-learning: Synergistic integration of LangChain, GPT models, and retrieval augmented generation (RAG)," in *International Conference on Smart Applications and Data Analysis*, Springer Nature Switzerland, 2024.

[7] V. Authors, "A comprehensive survey and guide to multimodal large language models in vision-language tasks," *arXiv preprint arXiv:2411.06284*, 2024.

[8] C. Burgan, J. Kowalski, and W. Liao, "Developing a retrieval augmented generation (RAG) chatbot app using adaptive large language models (LLM) and LangChain framework," *Proceedings of the West Virginia Academy of Science*, vol. 96, no. 1, 2024.

[9] I. Toniomo, "flutter_tts: A flutter plugin for text to speech." https://pub.dev/packages/flutter_tts, 2023.

[10] V. Authors, "Personalized E-learning recommender system based on autoencoders," *Applied System Innovation*, vol. 6, no. 6, 2023.

[11] V. Authors, "A digital recommendation system for personalized learning to enhance online education: A review," *IEEE Xplore*, 2024.

[12] U. F. Ikwuanusi, P. A. Adepoju, and C. S. Odionu, "AI-driven solutions for personalized knowledge dissemination and inclusive library user experiences," *International Journal of Engineering Research Updates*, vol. 4, no. 2, pp. 052–062, 2023.